IBM

eserver

iSeries

CICS for iSeries Intercommunication

Version 5

SC41-5456-00

IBM

@server

iSeries

CICS for iSeries Intercommunication

Version 5

SC41-5456-00

Note

Before using this information and the product it supports, be sure to read the information in "Notices" on page 157.

First Edition (September 2002)

This edition applies to version 5, release 2, modification 0 of IBM CICS Transaction Server for iSeries (5722-DFH) and to all subsequent releases and modifications until otherwise indicated in new editions. This edition applies only to reduced instruction set computer (RISC) systems.

This edition replaces SC33-1388-01.

Contents

About CICS for iSeries Intercommunication (SC41-5456)

This book is about setting up a CICS® for iSeries® to communicate with another CICS for iSeries system, or with any other member of the CICS products:

- CICS on System/390
- CICS on OS/2
- CICS on Open Systems
- CICS Clients

Intercommunication between two different CICS products is called CICS inter-product communication. For an overview of CICS inter-product communication, see the *CICS Family: Interproduct Communication*, SC34-6030-00 manual, which explains the documentation scheme of which this book is a part. "CICS/400 library" on page ix lists the other books in that scheme.

Who should read this book

This book is for those responsible for planning and implementing the CICS/400 side of an intercommunication link between two CICS systems, including the case where both sides of the link are CICS/400 systems.

This book assumes some familiarity with CICS resource definition and application programming, and with the use of iSeries CL commands, iSeries work management, and configuration of iSeries for communication.

You should have a copy of the companion book *CICS Family: Interproduct Communication*, in which Part 1 introduces CICS family intercommunication, and Part 2 gives help in selecting an intercommunication function, followed by detailed discussions of each function.

You may need knowledge of or access to information about the remote CICS system with which you want to communicate.

Conventions and terminology used in this book

The following CICS products run on computers of the S/370™, System/390, or zSeries™ family and support communications 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 2, program number 5697-E93
- CICS Transaction Server for OS/390® Version 1, program number 5655-147
- CICS/ESA Version 4, program number 5655-018
- CICS Transaction Server for VSE/ESA™, program number 5648-054
- CICS/VSE Version 2, program number 5686-026

The term CICS for OS/2 refers to CICS Transaction Server for OS/2 Warp Version 4.1 (which contains CICS for OS/2 Version 3.1.)

The term CICS on Open Systems is used as a generic name for:

- TXSeries™ Version 5.0 for Multi-platforms, 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)

The terms **mainframe** or **System/390** are used to refer to any System/370, System/390, or zSeries computer on which one of the above products can run. The terms **nonmainframe** or **non-System/390** refer to the hardware platforms used by other CICS products; for example, iSeries (used by CICS/400®), IBM-compatible personal computers (used by CICS for OS2) and RISC System/6000® or pSeries™ (used by CICS on Open Systems).

In statements that apply to each of the mainframe products, the generic term **CICS/mainframe** is used to represent all of them. A particular CICS/mainframe product is referred to by name only if there is a difference in its interface to CICS/400 as compared with the interface from other CICS/mainframe products.

The notation **CICS/400–CICS OS/2** is used to refer to communication in either direction between CICS/400 and CICS OS/2. To specify communication in only one direction, an arrow is added. For example, CICS/mainframe–CICS/400 function shipping refers to function shipping from CICS/mainframe to CICS/400 or from CICS/400 to CICS/mainframe. CICS/400→CICS/mainframe function shipping refers only to function shipping from CICS/400 to CICS/mainframe.

To conform with both CICS documentation and OS/400 documentation, the keywords used in EXEC CICS commands are called options and the keywords used in CL commands are called parameters.

All OS/400 CL commands can be entered at an OS/400 terminal or issued by a CL program. Command parameters are documented and indexed by showing the scren field name followed by the program parameter name in parentheses, for example:

In the parameter description: Destination (DEST)
In the index: destination (DEST)

VTAM® refers to ACF/VTAM, the Virtual Telecommunications Access Method.

Prerequisite and related information

Use the iSeries Information Center as your starting point for looking up iSeries technical information.

You can access the Information Center two ways:

- From the following Web site:
  `http://www.ibm.com/eserver/iseries/infocenter`
- From CD-ROMs that ship with your Operating System/400 order:

iSeries Information Center, SK3T-4091-02. This package also includes the PDF versions of iSeries manuals, *iSeries Information Center: Supplemental Manuals*, SK3T-4092-01, which replaces the Softcopy Library CD-ROM.

The iSeries Information Center contains advisors and important topics such as Java™, TCP/IP, Web serving, secured networks, logical partitions, clustering, CL commands, and system application programming interfaces (APIs). It also includes links to related IBM Redbooks™ and Internet links to other IBM Web sites such as the Technical Studio and the IBM home page.

With every new hardware order, you receive the *iSeries Setup and Operations CD-ROM*, SK3T-4098-01. This CD-ROM contains IBM **@server** iSeries Access for Windows and the EZ-Setup wizard. iSeries Access offers a powerful set of client and server capabilities for connecting PCs to iSeries™ servers. The EZ-Setup wizard automates many of the iSeries setup tasks.

CICS/400 library

These books form the CICS/400 library that is delivered with the product:

CICS for iSeries Administration and Operations Guide, SC41-5455-00

This guide gives introductory information about CICS/400. It then provides information about system and resource definition, setup of a system, and operator commands.

CICS for iSeries Application Programming Guide, SC41-5454-01

This manual provides programming guidance information, in narrative form with examples. This is followed by the reference section describing the syntax and use of each command.

CICS for iSeries Intercommunication, SC41-5456-00

This manual describes the CICS/400 side of communication between CICS systems running on different platforms. There is a similar manual for each CICS platform.

CICS for iSeries Problem Determination, SC41-5453-00

This manual provides guidance in problem determination for users of CICS/400.

CICS Family: Interproduct Communication, SC34-6030-00

This manual, which is also part of the libraries of the other CICS family members, gives an overview of communication between CICS systems running on different platforms.

CICS Family: API Structure, SC33-1007-02

This manual, which is also part of the libraries of the other CICS family members, gives a quick reference to the level of support that each member of the CICS family gives to the CICS application programming interface. It is designed for customers and software vendors developing applications able to run on more than one CICS platform and porting applications from one platform to another.

Books from related libraries

Some other manuals may be useful to the readers of this book.

CICS intercommunication manuals

CICS on System/390 Intercommunication Guide (refer to the Intercommunication Guide for your CICS on System/390 product)

CICS for OS/2 Intercommunication

CICS on Open System Intercommunications Guide

iSeries manuals

Communications Configuration, SC41-5401-00

ICF Programming, SC41-5442-00

Communications Management, SC41-5406-02

APPC Programming, SC41-5443-00

Work Management, SC41-5306-03

Globalization topic in the iSeries Information Center

Software Installation, SC41-5120-06

Other books

SNA Formats, GA23-3136

SNA LU 6.2 Peer Protocols Reference , SC31-6808

CPI Communications Reference, SC26-4399

AIX SNA Server/6000 V2R1 User's Guide, SC31-7002

How to send your comments

Your feedback is important in helping to provide the most accurate and high-quality information. If you have any comments about this book or any other iSeries documentation, fill out the readers' comment form at the back of this book.

- If you prefer to send comments by mail, use the readers' comment form with the address that is printed on the back. If you are mailing a readers' comment form from a country other than the United States, you can give the form to the local IBM branch office or IBM representative for postage-paid mailing.
- If you prefer to send comments by FAX, use either of the following numbers:
  - United States, Canada, and Puerto Rico: 1-800-937-3430
  - Other countries: 1-507-253-5192
- If you prefer to send comments electronically, use one of these e-mail addresses:
  - Comments on books:

    RCHCLERK@us.ibm.com
  - Comments on the iSeries Information Center:

    RCHINFOC@us.ibm.com

Be sure to include the following:

- The name of the book or iSeries Information Center topic.
- The publication number of a book.
- The page number or topic of a book to which your comment applies.

Summary of Changes

This section summarizes the major changes made to the *CICS for iSeries Intercommunication* book for Version 5 Release 2.

Changes for this release includes support for connectivity of CICS Universal Clients and the CICS Transaction Gateway to CICS for iSeries using TCP/IP connectivity.

Part 1. Introduction

This part introduces CICS intercommunication.

Chapter 1. Overview

> **Note**
> In this chapter, and throughout this book, the generic term CICS/mainframe or CICS for OS/390 are used to represent products run on System/370, System/390 or zSeries platforms. See the About CICS® for iSeries Intercommunications section at the front of this book for a list of specific products.

This chapter describes the CICS/400* intercommunication facilities under the following headings:

Intercommunication functions

CICS family inter-product communications primarily use Advanced Program-to-Program Communication (APPC), part of IBM's Systems Network Architecture (SNA), as their communications protocol. Support for this protocol is normally provided by an operating system extension, such as Communications Manager/2 in OS/2, or directly by the operating system itself, as in OS/400. CICS/400 relies on the OS/400's Intersystem Communication Function (ICF) to provide the APPC interface that it requires, while CICS/400 resource definitions and programming interfaces offer the user a simple way of defining distributed resources, and writing distributed transactions.

CICS for iSeries supports all the functions discussed in this chapter links to any of the following products:

- Another CICS for iSeries or CICS/400 system
- CICS on OS/390
- CICS on Open Systems
- CICS for OS/2

Support is either at synchronization level 1 or 2, and in either direction between the systems.

Distributed program link (DPL)

Distributed program link (DPL) enables an application program in a local CICS system to issue an EXEC CICS LINK command to link to a program in a remote CICS system, which returns control to the calling program. Distributed program link:

- Provides a way to access data not maintained by CICS but accessed by a remote CICS system, for example, IMS™, DL/I, and SQL databases on a

CICS/mainframe system. Existing mainframe programs can be used to access the data. (Another way of accessing this data is to use distributed transaction processing, see 5.)

- Provides a way to access BDAM files on a mainframe CICS system. Existing mainframe programs can be used to access the data. (Another way of accessing this data is to use distributed transaction processing, see 5.)
- Provides improved performance for a distributed system. For example, a single link can achieve a data set browse that would require multiple flows if function shipping were used.
- Gives the CICS/400 programmer access to programs that cannot be ported from the mainframe system.
- Allows a CICS/400 programmer to use an APPC link without needing to know the protocol.

In CICS, the linked program runs under the mirror transaction (CPMI for CICS/400), using that transaction's attributes, for example, task priority, security attributes, and keys.

Function shipping

Function shipping enables CICS application programs to access CICS resources owned by a remote CICS system.

A function shipping request takes the form of a normal EXEC CICS command. If the SYSID parameter is used in the EXEC CICS command or the resource is defined as remote, the application-owning system recognizes that function shipping is required and ships the request to the remote resource-owning system.

APPC protocol

The mirror program, running in the resource-owning CICS system, handles incoming function shipping requests.

CICS/400 supports synchronization level 2. Synchronization of recoverable changes is achieved using the CICS APPC protocols, described in Chapter 12, "Syncpointing a distributed process" on page 121.

DL/I database access

CICS/400 cannot ship requests that directly access DL/I databases. To access a DL/I database owned by CICS/mainframe from CICS/400, use distributed transaction processing or distributed program link (DPL).

Data conversion

CICS for OS/2 and CICS for Open Systems use ASCII[1] data representation; CICS/400 and CICS/mainframe systems use EBCDIC[2]. Conversion of user data is performed as necessary in the resource-owning system. For example, for CICS OS/2→CICS/400 function shipping, CICS/400 converts the user data. For CICS/400→CICS OS/2 function shipping, CICS OS/2 converts the user data.

Transaction routing

Transaction routing enables a terminal in one CICS system to run a transaction in another CICS system. The usual way to initiate transaction routing is by entering a remote transaction ID at a local terminal. Other ways are:

1. American National Standard Code for Information Interchange.
2. Extended Binary-Coded Decimal Interchange Code.

- Automatic transaction initiation, see *CICS Family: Interproduct Communication*, SC34-6030-00.
- CRTE transaction, see the *CICS for iSeries Administration and Operations Guide*, SC41-5455-00.

Because CICS inter-product communication uses a 3270 data stream, transactions can be routed only from terminals on which a 3270 data stream can be displayed. 5250 terminals are 3270-compatible.

Automatic transaction initiation

Automatic transaction initiation (ATI) requests can be issued in either direction between mainframe and nonmainframe CICS systems. When the request is received by the terminal-owning system, processing proceeds in exactly the same manner as if an operator entered the transaction code at the terminal, that is, as for transaction routing. For a full discussion of ATI, see the *CICS Family: Interproduct Communication* manual.

Asynchronous processing

Asynchronous processing is a form of intercommunication in which one transaction initiates another, and the two transactions then run independently of each other, that is, asynchronously.

CICS supports asynchronous processing in two ways:

1. Use of the interval control commands EXEC CICS START and EXEC CICS RETRIEVE. This is a special case of function shipping, in which the shipped request is an EXEC CICS START command.

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

   A CICS transaction that is initiated by a remotely issued START request can use the EXEC CICS RETRIEVE command to retrieve any data associated with the request.
2. Use of distributed transaction processing (DTP).

   This is 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 enables you to send data directly and, optionally, to receive data from the remote transaction. Your transaction design determines the format and volume of the data you exchange. For example, you can use repeated EXEC CICS SEND commands to pass multirecord files. In terms of command sequencing, error recovery, and synchronization, it is full DTP.

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

Distributed transaction processing (DTP)

Distributed transaction processing (DTP) enables transactions running in one CICS system to initiate and communicate synchronously with transactions in another CICS system [3] The initiating transaction can be in a CICS/400 system or in a non-CICS/400 system. Synchronization level 2 is supported.

3. CICS/400 also supports DTP with non-CICS systems, and with partners that do not use the CICS API.

DTP is an alternative to DPL as a way for CICS/400 to access data not maintained by CICS but accessed by a remote CICS system.

Application programs can issue EXEC CICS commands for APPC conversations and so control the allocation and use of an APPC session. To do this, a program must be aware of the state of the conversation over the intersystem link at any given time.

The EXEC CICS commands used to control an APPC conversation are:

```
ALLOCATE, CONNECT PROCESS, EXTRACT PROCESS, SEND, RECEIVE, CONVERSE,
ISSUE CONFIRMATION, ISSUE ERROR, ISSUE ABEND, FREE, WAIT CONVID,
EXTRACT ATTRIBUTES, ISSUE SIGNAL, and ISSUE PREPARE
```

For reference information on these commands, see the *CICS for iSeries Application Programming Guide*. For guidance in their use, see Part 3, "Distributed transaction programming" on page 97 of this book.

Security

CICS/400 is an integral part of the OS/400 operating system, and uses the security provided by the OS/400 operating system. If a CICS/400 transaction attempts an intercommunication operation for which it is not authorized, OS/400 returns an error code. CICS/400 presents the error to the application as a CICS exception condition—for example, NOTAUTH.

For an overview of CICS/400 security and guidance in its use, see Chapter 8, "Security" on page 85.

Summary of CICS/400 intercommunication

Table 1 shows the communication functions that a CICS/400 product can support on all possible CICS inter-product links.

If a function is shown as supported in Table 1, data conversion, where necessary, is supported at each end of the link.

CICS/400 supports sync levels 0, 1, and 2. Sync levels are described in the *CICS Family: Interproduct Communication* manual.

Using distributed transaction programming, CICS/400 can communicate on an LU6.2 link with any product that supports APPC protocols. This includes non-CICS and non-IBM* products.

Table 1. Sync levels supported for CICS/400 — CICS communication. In the table, 2,1 means that sync level 2 is supported, unless single sessions are being used.

CICS/400	CICS/400	CICS for OS/390	CICS OS/2	CICS for Open Systems
Function shipping outbound	2,1	2,1	1	2,1
Function shipping inbound	2,1	2,1	1	2,1
Transaction routing outbound	1	1	1	1

Table 1. Sync levels supported for CICS/400 — CICS communication (continued). In the table, 2,1 means that sync level 2 is supported, unless single sessions are being used.

CICS/400	CICS/400	CICS for OS/390	CICS OS/2	CICS for Open Systems
Transaction routing inbound	2,1	2,1	1	2,1
Distributed program link outbound with SYNCONRETURN	1	1	1	1
Distributed program link outbound with no SYNCONRETURN	2,1	2,1	1	2,1
Distributed program link inbound with SYNCONRETURN	1	1 (see note 1)	1	1
Distributed program link inbound with no SYNCONRETURN	2,1	2,1	1	2,1
Distributed transaction processing initiated by CICS/400	2,1,0	2,1,0	1,0	2,1,0
Distributed transaction processing initiated by partner	2,1,0	2,1,0	1,0	2,1,0
Asynchronous processing inbound and outbound (see note 2)	2,1	2,1	2,1	2,1
Notes: 1. CICS/ESA 3.3, CICS/ESA 4.1, and CICS/VSE 2.2 only 2. Sync level depends on the options used with the EXEC CICS command.				

Part 2. Setup and Programming

This part gives guidance on configuring CICS/400 for intercommunication, and on the use of the CICS intercommunication functions, which are introduced in the *CICS Family: Interproduct Communication* book. For each of these functions, except distributed transaction processing, this part:

- Gives guidance for the application programmer and the end user
- Describes how to set up CICS/400 to support the function

Additional chapters describe how to set up intercommunication security and data conversion when it is required.

DTP needs no setup beyond the configuration of CICS/400 for intercommunication. The programming requirements are considerable and are described in Part 3, “Distributed transaction programming” on page 97.

The main descriptions of resource definition commands are in the *CICS for iSeries Administration and Operations Guide*. This part gives additional guidance on parameters that are relevant to intercommunication.

Chapter 2. Configuring CICS/400 for intercommunication

CICS/400 inter-product communication uses LU6.2 connections created and controlled by the OS/400 intersystem communication function (ICF).

There are three main stages in the configuration of a CICS/400 system to communicate with another system:

1. Setting up the OS/400 communications entries, described 14
2. Creating an OS/400 subsystem that defines the CICS/400 system to OS/400, described 26
3. Setting up the CICS resource definitions, described 39

Figure 1 on page 12 shows these three steps. This chapter starts by describing the OS/400 definitions, shown at the bottom of Figure 1 and works up through the subsystem definitions shown in the middle of the figure, to the CICS definitions shown at the top of the figure.

This chapter concludes with a description on 46 of the definitions needed to configure a system for intrasystem communication, that is communication between two subsystems on the same OS/400.

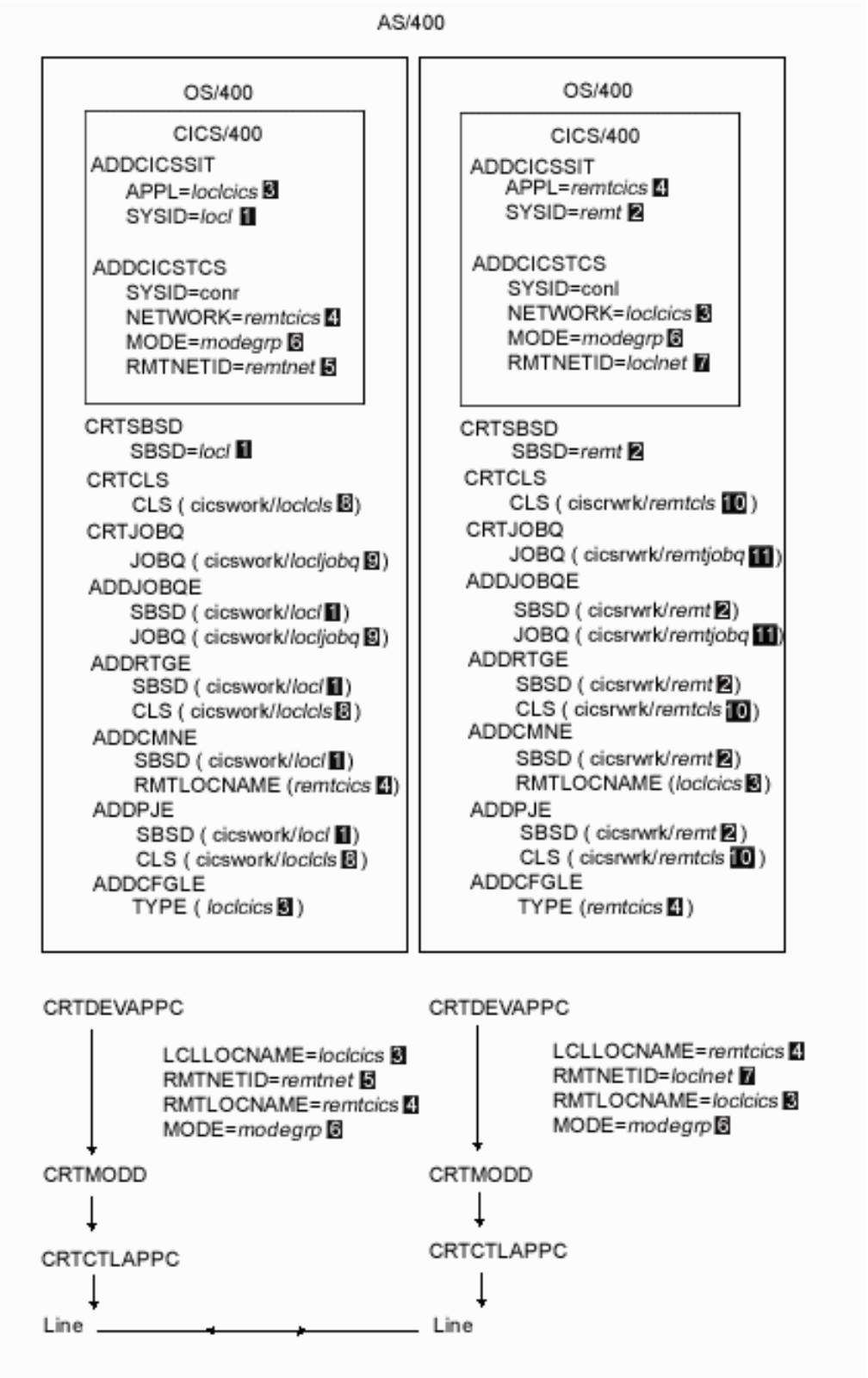


Figure 1. CICS/400 intercommunication configuration commands

Notes on figure:

1. This figure shows, at the bottom, the commands used to set up OS/400 definitions, in the middle, the commands used to set up an OS/400 subsystem, and at the top, the commands for setting up CICS resource definitions for intercommunication.
2. The figure represents two CICS/400 systems linked by an APPC connection. Values flagged with the same number must match. The numbers represent:

   1 LOCL - local CICS system identifier

   2 REMT - remote CICS system identifier

   3 LOCLCICS - local CICS application identifier (APPLID)

   4 REMTCICS - remote CICS application identifier (APPLID)

   5 REMTNET - remote system network identifier

   6 MODEGRP - mode used for intercommunication

   7 LOCLNET - local system network identifier

   8 LOCLCLS - local class

   9 LOCLJOBQ - local job queue

   10 REMTCLS - remote class

   11 REMTJOBQ - remote job queue

The SYSID value in the ADDCICSTCS command is used in each remote resource definition to specify the location of the resource. This is shown in the definitions in Chapter 4 through Chapter 7. As Figure 1 on page 12 indicates, the SYSID value in ADDCICSTCS does not have to match any other value. It can be helpful to make it match the remote subsystem name, making the entries SYSID=remt and SYSID=locl respectively.

Parameters shown in examples of CL commands in this chapter are given the prefix LOCL and REMT to indicate that they belong to the local and remote systems respectively. (These prefixes are intended for usability, and there are no system constraints on parameter values except that they should match where specified in Figure 1 on page 12.)

The commands described in this chapter are summarized in Table 2.

Table 2. Commands described in this chapter

Command	*Purpose*
CRTLINxxxx CRTCTLxxxx CRTMODD CRTDEVxxxx	Commands for setting up OS/400 communication entries. xxxx is the type of link, for example: SDLC - synchronous data link control; TRN - token ring; APPC - advanced program-to-program communication
CRTSBSD CRTCLS CRTJOBQ ADDJOBQE	Commands for creating OS/400 subsystem entries
ADDRTGE ADDCMNE ADDPJE ADDCFGLE	Commands for adding entries to OS/400 subsystems

Table 2. Commands described in this chapter (continued)

Command	*Purpose*
ADDCICSTCS ADDCICSSIT	Commands for setting up CICS resource definitions

Unless otherwise stated, all the commands mentioned in this chapter are fully described in the *Work Management* book. This chapter shows how to link the CICS/400 definitions to the OS/400 definitions. For a full description of the OS/400 definitions, see the *Communications Configuration* book.

Setting up OS/400 communications objects

This section describes the commands that create OS/400 resource definitions. Like all OS/400 CL commands, these commands can be issued from an OS/400 command line[4], an HLL application, or a CL program. CICS/400 requires the same basic OS/400 configuration objects that are used by most OS/400 communication configurations. Because this book is for CICS/400 users only, the explanations below relate solely to CICS systems and ignore the existence of non-CICS systems.

The required OS/400 objects are:

Line description
A line description defines a physical communication link between the local CICS/400 system and one or more remote CICS systems. The description defines the physical interface and communication protocol used by the line.

Lines are usually defined as part of the iSeries hardware. There is a command, CRTLINxxxx, which you could use if necessary.

For each CICS system with which the local CICS/400 system is to communicate across the line, a separate controller description is required.

Controller description
A controller description is required for each controller on the system, that is, one for each remote iSeries or VTAM domain that attaches to the system via communication links. The controller description describes the adjacent system.

A controller corresponds roughly to an SNA **physical unit**.

The command used to create a controller definition is described on 16.

Mode description
A mode is a set of attributes to be associated with a connection. These attributes relate to the number of sessions and conversations, types of session, pacing values, and the number of request units. Mode descriptions are referred to in device descriptions, and also in CICS/400 remote system definitions.

The command used to create a mode definition is described 22.

Device description
A device description is required for each remote CICS system that is to communicate with the local CICS/400 system. The description includes

4. CICS/400 users can switch to the OS/400 main menu from the CEDA transaction.

addressing information and detailed data about communication between the local CICS system and the specific remote system.

Because several CICS systems can be located in a single remote OS/400 or VTAM domain, several device descriptions can be associated with a single controller description.

A device corresponds roughly to an SNA **logical unit**. A device description is roughly the OS/400 equivalent of the CICS/400 TCS entry.

The command used to create a device definition is described 24.

How the commands are described

The description of each command shows the OS/400 screen for online command entry, and gives field descriptions related to that screen. Each field description is headed by its screen name with, in parentheses, the name used to refer to the field in a program.

CL command defaults

The defaults given in the CL command descriptions are those that are supplied with the OS/400 system. You should check that your installation has not made any changes to these command default parameters.

In some cases, fields are displayed only if a specific value is entered in an associated field. For example, in the CRTCTLAPPC screen, four fields are displayed until a value is entered in the **Link type** field. For CICS/400, you might enter a link type of *SDLC–the screen then displays all the additional fields required for an SDLC link.

Before a completed definition is available for use, it must be installed into the OS/400 runtime resource table definitions. You can install a definition with the INSCICSGRP command from the OS/400 main menu screen shown in Figure 2 on page 16.

```
MAIN                          OS/400 Main Menu
                                                        System:   WINAS5
Select one of the following:

     1. User tasks
     2. Office tasks
     3. General system tasks
     4. Files, libraries, and folders
     5. Programming
     6. Communication
     7. Define or change the system
     8. Problem handling
     9. Display a menu
    10. Information Assistant options
    11. PC Support tasks

    90. Sign off

Selection or command
===> inscicsgrp

F3=Exit   F4=Prompt   F9=Retrieve   F12=Cancel   F13=User support
F23=Set initial menu
```

Figure 2. The OS/400 main menu

OS/400 controller description (APPC), CRTCTLAPPC

Figure 3 shows the OS/400 screens for setting up controller descriptions using the CRTCTLAPPC command.

```
                    Create Ctl Desc (APPC) (CRTCTLAPPC)

Type choices, press Enter.

Controller description . . . . .           Name
Link type  . . . . . . . . . . . > *SDLC    *IDLC, *LAN, *LOCAL, *SDLC...
Online at IPL  . . . . . . . . .   *YES     *YES, *NO
Switched connection  . . . . . .   *NO      *NO, *YES
Switched network backup  . . . .   *NO      *NO, *YES
APPN-capable . . . . . . . . . .   *YES     *YES, *NO
Attached nonswitched line  . . .           Name
Maximum frame size . . . . . . .           265-16393, 256, 265, 512...
Remote network identifier  . . .   *NETATR  Name, *NETATR, *NONE, *ANY
Remote control point . . . . . .           Name, *ANY
Exchange identifier  . . . . . .           00000000-FFFFFFFF
Data link role . . . . . . . . .   *NEG     *NEG, *PRI, *SEC
Station address  . . . . . . . .           00-FE
APPN CP session support  . . . .   *YES     *YES, *NO
APPN node type . . . . . . . . .   *ENDNODE *ENDNODE, *LENNODE...
APPN transmission group number     1        1-20, *CALC
                                                                More...
F3=Exit  F4=Prompt  F5=Refresh   F10=Additional parameters   F12=Cancel
F13=How to use this display      F24=More keys
Parameter CTLD required.
```

```
                    Create Ctl Desc (APPC) (CRTCTLAPPC)

Type choices, press Enter.

Autodelete device  . . . . . .   1440          1-10000, *NO
User-defined 1 . . . . . . . .   *LIND         0-255, *LIND
User-defined 2 . . . . . . . .   *LIND         0-255, *LIND
User-defined 3 . . . . . . . .   *LIND         0-255, *LIND
Text 'description' . . . . . .   *BLANK

                                                                Bottom
F3=Exit  F4=Prompt  F5=Refresh   F10=Additional parameters   F12=Cancel
F13=How to use this display      F24=More keys
```

Figure 3. The CRTCTLAPPC command screens

The parameters on the CRTCLTAPPC command screen are as follows:

Controller description (CTLD)
The name of the controller description.

Link type (LINKTYPE)
The type of line to which this controller is attached. For example, you might enter *SDLC, representing a synchronous data link control (SDLC) line.

Online at IPL (ONLINE)
Specifies whether this object is automatically varied on at initial program load (IPL). The possible values are:

***YES**
The controller is automatically varied on at IPL.

***NO**
The controller is not automatically varied on at IPL.

Switched connection (SWITCHED)
Specifies whether this controller is attached to a switched line. The possible values are:

***NO**
This controller is attached to a nonswitched line. Specify this value for controllers attaching to an X.25 permanent virtual circuit (PVC).

***YES**
This controller is attached to a switched line. Specify this value for controllers attached to an X.25 switched virtual circuit (SVC). Also specify this value for controllers attached to a local area network.

Switched network backup (SNBU)
Specifies whether the remote system modem has the switched network backup (SNBU) feature. The backup feature is used to bypass a broken nonswitched (leased line) connection by establishing a switched connection. To activate SNBU, you must change the controller description of the modem from nonswitched to switched by specifying *YES for the Activate switched network backup (ACTSNBU) parameter.

Note: If the modem model you are using is an IBM 386x, 586x, or 786x, you should not change the controller description. Instead, manually switch the modem to the nonswitched mode, and manually dial the connection.

The possible values are:

***NO**
The remote system modem does not have the SNBU feature.

***YES**
The remote system modem has the SNBU feature.

APPN-capable (APPN)
Specifies whether this controller is for Advanced Peer-to-Peer Networking® (APPN®). The possible values are:

***YES**
This controller is for APPN.

***NO**
This controller is not for APPN.

Attached nonswitched line (LINE)
The name of the nonswitched line to which this controller is attached. A description of the line must already exist.

Note: Specify this parameter for controllers attaching to an X.25 permanent virtual circuit (PVC).

Maximum frame size (MAXFRAME)
The maximum frame size the controller can send or receive. For an SDLC link, specify 265, 521, 1033, or 2057.

Note: These values are valid only if TYPE(*BLANK) is specified when the controller is created.

Remote network identifier (RMTNETID)
Specifies how to determine the name of the remote network in which the adjacent control point resides. The possible values are:

***NETATR**
The LCLNETID value specified in the system network attributes is used.

***NONE**
No remote network identifier (ID) is used.

***ANY**
The system determines which remote network identifier is used.

remote-network-identifier
The remote network identifier.

Remote control point (RMTCPNAME)
Specifies the name of the remote control point or indicates that the system is to determine the name. The possible values are:

***ANY**
The system determines the name of the remote control point used.

remote-control-point-name
The remote control point name.

Exchange identifier (EXCHID)
The remote exchange identifier of this controller. The controller sends (exchanges) its identifier to another location when a connection is established. The 8-digit hexadecimal identifier contains 3 digits for the block number and 5 digits for the identifier of the specific controller.

The block number must have the following value depending on the remote system or controller: 5251 = 020; 5294 = 045; 5394 = 05F; 3694 = 02F; 4701 or 4702 = 057; 3651, 3684 or 4684 = 005; 4680 = 04D; 3601 (configured as a 4701) = 016; 3174 or 3274 = between 001 and 0FE; S/36 = 03E; S/38 = 022; Displaywriter = 03A; OS/400 = 056. In addition, the last 5 digits of the exchange identifier must begin with 000 for 5251, 5394, and 5294 controllers.

Data link role (ROLE)
Specifies the data link role that the remote controller has on this connection. The primary station is the controlling station and the secondary station is the responding station. The primary station controls the data link by sending commands to the secondary station, and the secondary station responds to the commands.

The possible values are:

***NEG**
The local and remote systems negotiate which is the primary station on this connection.

***PRI**
The remote system is the primary station.

***SEC**
The remote system is a secondary station.

Station address (STNADR)
The station address used when communicating with the controller. Valid values for an APPC SDLC controller are in the hexadecimal range 01 through FE.

APPN CP session support (CPSSN)
Specifies whether this controller supports sessions between control points. The possible values are:

***YES**
This controller supports sessions between control points.

***NO**
This controller does not support sessions between control points.

APPN node type (NODETYPE)
Specifies the type of APPN node that this controller represents. The possible values are:

***ENDNODE**
This node is an end node in an APPN network.

***NETNODE**
This node is a network node in an APPN network.

***LENNODE**
This node is a low-entry networking node in an APPN network.

***CALC**
The system determines the type of node that this controller represents.

APPN transmission group number (TMSGRPNBR)
Specifies the transmission group number for this controller, or indicates that the system should specify this number. The possible values are:

***CALC**
The system specifies the transmission group number.

transmission-group-number
The transmission-group-number. The value must be a value in the range 1 through 20. The default transmission group number is 1.

Autodelete device (AUTODLTDEV)
Specifies (1) whether the system should vary off and delete an automatically created device for this controller when its last session is unbound, and (2), if it should vary off and delete such a device, how long to wait before doing so. The possible values are:

1440
The system automatically varies off and deletes an automatically-configured idle device description after 1440 minutes (24 hours).

***NO**
The system does not automatically vary off and delete an automatically-configured idle device description.

wait-time
The number of minutes to wait before deleting an automatically-configured idle device description. Valid values are in the range 1 through 10 000.

User-defined x (USRDFNx)
Describes unique characteristics of the line that you want to control. These parameters are valid only if advanced peer-to-peer networking (APPN) is used on the system. The possible values are:

***LIND**
The user-defined value in the line description is used.

user-defined-x
A value ranging from 0 through 255.

Text 'description' (TEXT)
Provided for user-documentation purposes. The possible values are:

***BLANK**
No text is specified.

description
Text that briefly describes the object. Enter no more than 50 characters of text, enclosed in apostrophes.

OS/400 mode description, CRTMODD

Figure 4 shows the screen for setting up OS/400 mode descriptions.

```
                     Create Mode Description (CRTMODD)

Type choices, press Enter.

Mode description . . . . . .                Name
Maximum sessions . . . . . .   8            1-512
Maximum conversations  . . . .   8          1-512
Locally controlled sessions  . .   4        0-512
Pre-established sessions . . . .   0        0-512
Inbound pacing value . . . . . .   7         0-63
Outbound pacing value  . . . . .   7         0-63
Maximum length of request unit     *CALC     241-16384, *CALC
Text 'description' . . . . . .   *BLANK

Class-of-Service . . . . . .   #CONNECT      Name, #CONNECT, #BATCH




                                                                  Bottom
F3=Exit  F4=Prompt  F5=Refresh   F10=Additional parameters   F12=Cancel
F13=How to use this display      F24=More keys
```

Figure 4. The CRTMODD command screen

The parameters on the CRTMODD command screen are as follows:

Mode description (MODD)
The name of this mode description.

Maximum sessions (MAXSSN)
The maximum number of active sessions that are established for this mode. This number must be greater than or equal to the sum of the locally controlled sessions (LCLCTLSSN) value in this mode description and the number of locally controlled sessions specified at the remote location.

Valid values are in the range 1 through 512.

Maximum conversations (MAXCNV)
The maximum number of conversations that can be established at the same time with the remote system. The maximum number of conversations is the sum of synchronous and asynchronous conversations; this value must be greater than or equal to the value specified by the maximum sessions (MAXSSN) parameter.

In this description, a synchronous conversation is a conversation in which the source and the target programs are communicating with each other (in CICS terminology, this is called distributed transaction processing). An asynchronous conversation is a conversation in which the source program has detached itself from the conversation, but data is still being read by the target program (in CICS terminology, this is called asynchronous processing).

Valid values are in the range 1 through 512.

Locally controlled sessions (LCLCTLSSN)
The minimum number of locally controlled sessions that must be active to establish this mode. This value must be less than or equal to the value specified in the maximum sessions (MAXSSN) parameter.

Valid values are in the range 0 through 512.

Pre-established sessions (PREESTSSN)
The maximum number of locally controlled concurrent sessions that are established when the mode is started. Additional sessions are established as required, up to the maximum number of locally controlled sessions specified in the maximum sessions (MAXSSN) parameter. This value must be less than or equal to the value specified in the locally controlled sessions (LCLCTLSSN) parameter.

Valid values are in the range 0 through 512.

Inbound pacing value (INPACING)
The SNA pacing value used to schedule the incoming request/response units.

Valid values are in the range 0 through 63.

Outbound pacing value (OUTPACING)
The SNA pacing value used for outgoing request/response units.

Valid values are in the range 0 through 63.

Maximum length of request unit (MAXLENRU)
The maximum request unit (RU) length allowed. If you specify *CALC, the system calculates the value to use.

Valid values are *CALC or a number in the range 241 through 16384. *CALC is the recommended value. Common values for different types of line are:

SDLC 256, 512, 1024, 2048

Token-Ring Network
256, 512, 1024, 1985

X.25 (QLLC)
247, 503, 1015

X.25 (ELLC)
241, 497, 1009

For further information, see the *APPC Programming* book.

Text 'description' (TEXT)
Provided for user-documentation purposes. The possible values are:

***BLANK**
No text is specified.

description
Text that briefly describes the object. Enter no more than 50 characters of text, enclosed in single quotes.

Class-of-service (COS)
The path control network characteristics used by APPN. The possible values are:

#CONNECT
A class-of service definition provided by IBM.

#BATCH
A class-of service definition provided by IBM.

name
The name of a class-of service definition that you have created using the CRTCOSD command.

OS/400 device description (APPC), CRTDEVAPPC

If you are setting up communications with another system you need to specify only 1 device. For intrasystem communication you need to specify 2 devices; one for inbound communication, and one for outbound communication. These 2 devices must, of course, be on the same machine.

Figure 5 shows the screen for setting up device descriptions.

```
                  Create Device Desc (APPC) (CRTDEVAPPC)

Type choices, press Enter.

Device description . . . . . .               Name
Remote location  . . . . . .                 Name
Online at IPL  . . . . . .   *YES           *YES, *NO
Local location . . . . . .   *NETATR        Name, *NETATR
Remote network identifier  . . .  *NETATR   Name, *NETATR, *NONE
Attached controller  . . . . . .             Name
Mode . . . . . .             *NETATR        Name, *NETATR
             + for more values
Message queue  . . . . . .   QSYSOPR        Name, QSYSOPR
  Library  . . . . . .         *LIBL        Name, *LIBL, *CURLIB
APPN-capable . . . . . .     *YES           *YES, *NO
Single session:
  Single session capable . . . .  *NO        *NO, *YES
  Number of conversations  . . .             1-512


                                                                  Bottom
  F3=Exit  F4=Prompt  F5=Refresh   F10=Additional parameters   F12=Cancel
F13=How to use this display      F24=More keys
                                                                      +
```

Figure 5. The CRTDEVAPPC command screen

The parameters on the CRTDEVAPPC command screen are:

Device description (DEVD)
The name of the device description.

Remote location (RMTLOCNAME)
The name of the remote location with which your program communicates.

Online at IPL (ONLINE)
Specifies whether this object is automatically varied on at initial program load (IPL). The possible values are:

***YES**
This device is varied on automatically at IPL.

***NO**
This device is not varied on automatically at IPL.

Local location (LCLLOCNAME)
Specifies the unique location name that identifies the local system to remote devices. The name cannot be the same as that specified for the Remote location name (RMTLOCNAME) parameter. If the values specified on the Remote network ID and Local network ID parameters are the same, the combination of the names specified for the Local location name (LCLLOCNAME) and the Remote location name (RMTLOCNAME) parameters must be unique for each device description attached to the same controller.

The possible values are:

***NETATR**
The remote network identifier specified in the network attributes is used. Use the Display Network Attributes (DSPNETA) command to determine the default local location name.

local-location-name
The name (8 characters maximum) by which the local system is known to the remote device.

Remote network identifier (RMTNETID)
The name of the remote network. The possible values are:

***NETATR**
The remote network identifier specified in the network attributes is used.

***NONE**
The remote network name is X'40'.

remote-network-ID
The 8-character remote network name.

Attached controller (CTL)
The name of the controller to which this device is attached.

Note: To use this device for communicating with a remote location that resides on the same system as the local location, specify a controller description that was created with LINKTYPE (*LOCAL).

Mode (MODE)
The names of the modes that define the sessions on this device.

You can enter up to 14 mode names. If you are on an entry display and you need additional entry fields to enter these multiple values, type a plus sign (+) in the entry field on the line labeled "+ for more" and press Enter.

mode-name
Specify the name of mode descriptions used by this device. The mode name cannot be CPSVCMG or SNASVCMG because these mode names are reserved for system use.

Message queue (MSGQ)
The qualified name of the message queue to which operational messages for this device are sent.

Library
Specifies how to locate the named message queue. The possible values are:

***LIBL**
Search the library list to locate the message queue.

***CURLIB**
Search the current job library list to locate the message queue. If no current library entry exists in the library list, the QGPL is used.

library-name
The library in which the message queue is located.

QSYSOPR
Messages are sent to the QSYSOPR message queue.

APPN-capable (APPN)
Specifies whether this device is for Advanced Peer-to-Peer Networking (APPN). The possible values are:

***YES**
This device is for APPN.

***NO**
This device is not for APPN.

Single session (SNGSSN)
Specifies whether single or multiple sessions are used with remote locations. If single sessions are used, the number of conversations must be specified.

Single session capable
The possible values are:

***NO**
Multiple sessions are used.

***YES**
Single sessions are used.

Number of conversations
A number in the range 1 through 512. The default number of conversations is 10.

When you send a bind, the device is dynamically allocated for you with a generated name. You must set up a communications entry on your system that matches the device name.

Creating an OS/400 subsystem

Having set up your OS/400 communications entries, you need to create an OS/400 subsystem to enable inbound intersystem communication (ISC) requests to be routed to a local CICS/400 system.

The name of this subsystem must match the 4-character CICS system identifier of the CICS/400 system, that is, the name of the CICS control region.

The command used to create a subsystem is CRTSBSD. For example:

```
CRTSBSD SBSD(CICSWORK/LOCL) POOL((1 *BASE))
   TEXT('ISC requests for CICS Control Region LOCL')
```

You could specify the subsystem name to be "*" so that it is generic.

To enable the subsystem to process OS/400 batch job requests, an OS/400 class description, job queue description, and job queue entry are required. The commands used to set up these descriptions are CRTCLS, CRTJOBQ, and ADDJOBQE. For example:

```
CRTCLS CLS(CICSWORK/LOCLCLS) TEXT('Class for CICS Control Region LOCL')

CRTJOBQ JOBQ(CICSWORK/LOCLJOBQ)
   TEXT('Job queue for CICS Control Region LOCL')

ADDJOBQE SBSD(CICSWORK/LOCL) JOBQ(CICSWORK/LOCLJOBQ) MAXACT(5)
```

The job queue entry defines the queue for jobs allocated to run in this subsystem. The default value for the MAXACT parameter is 1. Therefore, you must include this parameter if you want more than one job to run in this job queue.

When you start the subsystem, take care to coordinate its commencement with that of the OS/400 supplied subsystem QCMN, otherwise QCMN will take in all APPC requests and your CICS/400 ISC requests will not reach the CICS subsystem and will fail. There are two ways around this problem; either you can start the CICS/400 subsystem before the QCMN subsystem on system startup, or after bringing up the required CICS/400 subsystem, you can vary off and on the *APPC devices for the CICS/400 connections. Provided that the correct communication entries have been added to the CICS/400 subsystem, this will cause OS/400 to allocate the *APPC devices to the CICS/400 subsystem, rather than QCMN.

Adding subsystem entries

To complement the subsystem definition, additions must be made to the following OS/400 components:

- Routing entries, command ADDRTGE
- Communications entries, command ADDCMNE
- Prestart Job entries, command ADDPJE
- Configuration list entries, command ADDCFGLE

Full details of these components are in the *Work Management* book. Note that all the examples in this section include the parameter SBSD(CICSWORK/LOCL) to link the entry with the correct subsystem.

Adding routing entries

To enable inbound ISC requests to be processed by CICS/400, a routing entry is required for the CICS/400 ISC program, AEGISICC. For example:

```
ADDRTGE SBSD(CICSWORK/LOCL) SEQNBR(100)
        CMPVAL('PGMEVOKE' 29) PGM(QCICS/AEGISICC)
        CLS(CICSWORK/LOCLCLS)
```

In the above example, the CMPVAL and PGM parameters must be coded exactly as shown. If the CICS control region also uses this subsystem, a second routing entry is necessary.

```
ADDRTGE SBSD(CICSWORK/LOCL) SEQNBR(9999)
        CMPVAL(*ANY) PGM(QSYS/QCMD)
        CLS(CICSWORK/LOCLCLS)
```

The ADDRTGE command screen is shown in Figure 6.

```
                        Add Routing Entry (ADDRTGE)

Type choices, press Enter.

Subsystem description  . . . . .               Name
  Library  . . . . . . . . . . .     *LIBL      Name, *LIBL, *CURLIB
Routing entry sequence number  .               1-9999
Comparison data:
  Compare value  . . . . . . . .

  Starting position  . . . . . .   1           1-80
Program to call  . . . . . . . .               Name, *RTGDTA
  Library  . . . . . . . . . . .     *LIBL      Name, *LIBL, *CURLIB
Class  . . . . . . . . . . . . .   *SBSD        Name, *SBSD
  Library  . . . . . . . . . . .               Name, *LIBL, *CURLIB
Maximum active routing steps . .   *NOMAX       0-1000, *NOMAX
Storage pool identifier  . . . .   1           1-10



                                                                    Bottom
F3=Exit  F4=Prompt  F5=Refresh  F12=Cancel  F13=How to use this display
F24=More keys
Parameter SBSD required.                                             +
```

Figure 6. The ADDRTGE command screen

The parameters on the ADDRTGE command screen are as follows:

Subsystem description (SBSD)
Specifies the name and library of the subsystem description to which the routing entry is added.

The possible library values are:

***LIBL**
The library list is used to locate the object.

***CURLIB**
The current library for the job is used to locate the object. If no library is specified as the current library for the job, QGPL is used.

library-name
Specify the name of the library where the object is located.

Routing entry sequence number (SEQNBR)
The sequence number of the routing entry that is added or changed. Routing data is matched against the routing entry compare values in ascending sequence number order. Searching ends when a match occurs or the last routing entry is reached. Therefore, if more than one match possibility exists, only the first match is processed.

Enter a unique sequence number in the range 1 through 9999 that identifies the routing entry.

Comparison data (CMPVAL)
Specifies a value that is compared with the routing data to determine whether this routing entry is used for starting a routing step for the job. If the routing data matches the routing entry compare value, that routing entry is used. Also, a starting position in the starting data character string can be used to specify the starting position in the routing data for comparison against the routing entry compare value.

The possible values are:

***ANY**
Any routing data is considered a match. If you specify *ANY, the routing entry must have the highest sequence number value of any routing entry in the subsystem description.

compare-value
Specify a value (any character string not exceeding 80 characters) that is compared with routing data for a match. When a match occurs, this routing entry is used to start a routing step.

A starting position in the routing data character string can be specified for the comparison; if no position is specified, position 1 is assumed. Specify a value in the range 1 through 80. Between the starting position and the end of the routing data, there must be at least as many characters as there are in the comparison value.

Program to call (PGM)
Specifies the name and library of the program called as the (first) program run in the routing step. No parameters can be passed to the specified program. The program name can be either explicitly specified in the routing entry, or extracted from the routing data. If a program name is specified in a routing entry, selection of that routing entry results in the routing entry program being called (regardless of the program name passed in an EVOKE function). If the program specified in the EVOKE function is called, *RTGDTA must be

specified. If the program does not exist when the routing entry is added or changed, a library qualifier must be specified because the qualified program name is kept in the subsystem description.

The possible values are:

***RTGDTA**
The program name is taken from the routing data that was supplied and matched against this entry. A qualified program name is taken from the routing data in the following manner; the program name is taken from positions 37 through 46, and the library name is taken from positions 47 through 56. Care should be used to ensure that routing entries that specify *RTGDTA are selected only for EVOKE functions on jobs that have specified the program name in the correct position in the routing data.

program-name
Specify the name and library of the program that is run from this routing entry.

The possible library values are:

***LIBL**
The library list is used to locate the named program.

***CURLIB**
The current library for the job is used to locate the named program. If no library is specified as the current library for the job, QGPL is used.

library-name
Specify the library where the named program is located.

Class (CLS)
Specifies the name and library of the class used for the routing steps started through this routing entry. The class defines the attributes of the routing step's running environment. If the class does not exist when the routing entry is added, a library qualifier must be specified because the qualified class name is kept in the subsystem description. The possible values are:

***SBSD**
The class having the same qualified name as the subsystem description, specified on the Subsystem description (SBSD) parameter is used for routing steps started through this entry.

class-name
Specify the name and library of the class that is used for routing steps started through this routing entry. The possible library values are:

***LIBL**
The library list is used to locate the class.

***CURLIB**
The current library for the job is used to locate the class. If no library is specified as the current library for the job, QGPL is used.

library-name
Specify the library in which the class is located.

Maximum active routing steps (MAXACT)
Specifies the maximum number of routing steps (jobs) that can be active at the same time through this routing entry. In a job, only one routing step is active at a time. When a subsystem is active and the maximum number of routing steps is reached, any subsequent attempt to start a routing step through this

routing entry fails. The job that attempted to start the routing step is ended, and a message is sent by the subsystem to the job log.

The possible values are:

***NOMAX**
There is no maximum number of routing steps that can be active at the same time and processed through this routing entry. This value is normally used when there is no reason to control the number of routing steps.

maximum-active-jobs
Specify the maximum number of routing steps that can be active at the same time through this routing entry. If a routing step being started would exceed this number, the job is implicitly ended.

Storage pool identifier (POOLID)
Specifies the pool identifier of the storage pool in which the program runs. The pool identifier specified here relates to the storage pools in the subsystem description. The possible values are:

1 Storage pool 1 of this subsystem is the pool in which the program runs.

pool-identifier
Specify the identifier of the storage pool defined for this subsystem in which the program runs. Valid values are in the range 1 through 10.

Adding communications entries

A communications entry is needed for each remote location or device with which this CICS system communicates. This example enables subsystem LOCL to communicate with remote location REMTLOC. The RMTLOCNAME parameter must match the NETWORK parameter of the ADDCICSTCS command (see 40).

```
ADDCMNE SBSD(CICSWORK/LOCL) RMTLOCNAME(REMTLOC) DFTUSR(USERID)
```

You may wish to add a DFTUSR parameter to the ADDCMNE command. This parameter defines the user profile under which to run the program START request. See “User security” on page 85. Note that the DFTUSR parameter defaults to *NONE. The supplied transaction CRTE does not send a user ID and will not work without a default user in the communications entry. For details of CRTE, see the *CICS for iSeries Administration and Operations Guide*.

The ADDCMNE command screen is shown in Figure 7 on page 32.

ADDCMNE

```
                    Add Communications Entry (ADDCMNE)

Type choices, press Enter.

Subsystem description  . . . . .               Name
  Library  . . . . . . . . . . .     *LIBL      Name, *LIBL, *CURLIB
Device . . . . . . . . . . . . .               Name, generic*, *ALL...
Remote location  . . . . . . . .               Name
Job description  . . . . . . . .   *USRPRF      Name, *USRPRF, *SBSD
  Library  . . . . . . . . . . .               Name, *LIBL, *CURLIB
Default user profile . . . . . .   *NONE        Name, *NONE, *SYS
Mode . . . . . . . . . . . . . .   *ANY         Name, *ANY
Maximum active jobs  . . . . . .   *NOMAX       0-1000, *NOMAX









                                                                  Bottom
F3=Exit  F4=Prompt  F5=Refresh  F12=Cancel  F13=How to use this display
F24=More keys
Parameter SBSD required.
```

Figure 7. The ADDCMNE command screen

The parameters on the ADDCMNE command screen are as follows:

Subsystem description (SBSD)
Specifies the name and library of the subsystem description to which the communications entry is being added or in which it is being changed.

The possible library values are:

***LIBL**
The library list is used to locate the object.

***CURLIB**
The current library for the job is used to locate the object. If no library is specified as the current library for the job, QGPL is used.

library-name
The name of the library where the object is located.

Device (DEV)
Specifies the name or type of the device used with this communications entry.

Note: You must specify a value on either this option or the Remote location (RMTLOCNAME) option, but not both.

The possible values are:

device-name
The name of the device description (CRTDEVAPPC) used with this communications entry.

generic*-device-name
The generic name of the device description used with this communications entry.

***ALL**
All communications device types or names are used with this communications entry.

***APPC**
All advanced program-to-program communications devices can be used with this communications entry. The devices created with the Create Device Description (APPC) (CRTDEVAPPC) command can be used.

***INTRA**
All INTRA communications devices can be used with this communications entry. The devices created with the Create Device (INTRA) (CRTDEVINTR) command can be used. This value is valid only when *ANY is specified on the Modeprompt (MODE) parameter.

***ASYNC, *BSCEL, *FINANCE, *RETAIL, *SNUF**
These device types are not supported by CICS/400.

Remote location (RMTLOCNAME)
The name of the remote location used with this communications entry. The remote location name specified in the associated Create Device Description (CRTDEVAPPC) command can be used here. No validity checking is done on the remote location name.

Note: You must specify a value for either this option or the Device (DEV) option, but not for both.

Job description (JOBD)
Specifies the name and library of the job description used for jobs that are started as a result of receiving a program START request, and processed through this communications entry. If the job description does not exist when the entry is added or changed, a library qualifier must be specified because the qualified job description name is kept in the subsystem description.

The possible values are:

***USRPRF**
The job description name that is specified in the user profile of the user that made the program START request is used for jobs that are processed through this communications entry.

***SBSD**
The job description having the same name as the subsystem description, specified on the subsystem description (SBSD) parameter, is used for jobs processed through this communications entry.

job-description-name
The name of the job description that is used for the jobs processed through this communications entry.

The possible library values are:

***LIBL**
The library list is used to locate the job description.

***CURLIB**
The current library for the job is used to locate the job description. If no library is specified as the current library for the job, QGPL is used.

library-name
The library where the job description is located.

Default user profile (DFTUSR)
Specifies the default user profile used for a program START request that

contains no security information. This user profile is not used for program START requests that contain a password or that specify a user profile (whether valid or invalid).

The possible values are:

***NONE**
No user profile is specified as the default.

***SYS**
All user program START requests will be treated the same as *NONE. For program START requests sent by system functions, the request will run under a predetermined user profile if a user profile is not specified on the program START request.

user-profile-name
Specify the name of the user profile that is used for all program START requests that enter the system through this communications entry and contain no security information.

Note: The names QSECOFR, QSPL, QDOC, QDBSHR, QRJE, and QSYS are not valid entries for this option.

Mode (MODE)
Specifies the mode name of the communications device or remote location name whose communications entry is being changed. The possible values are:

***ANY**
Any available modes defined to the communications device or remote location are allocated to the subsystem. If the communications device does not have defined modes associated with it, the communications device itself is allocated to the subsystem.

mode-name
The mode name of the communications device or remote location name that is being changed.

Maximum active jobs (MAXACT)
Specifies the maximum number of jobs (received program START requests) that can be active at the same time through this communications entry. The possible values are:

***NOMAX**
There is no maximum number of jobs that can be active at the same time through this communications entry.

maximum-active-jobs
The maximum number of jobs that can be active at the same time through this communications entry.

Adding prestart job entries

Prestart job entries can be used to accelerate the handling of inbound ISC requests. A number of inbound ISC jobs can be initiated when the CICS control region is started and so are ready to run without any delay. To make this happen, use the ADDPJE command to add a number of prestart jobs to the subsystem. For example:

```
ADDPJE SBSD(CICSWORK/LOCL) PGM(QCICS/AEGISICC)
       STRJOBS(*NO) MAXJOBS(5) CLS(CICSWORK/LOCLCLS) WAIT(*YES)
```

Notes:

1. The job class (CLS) parameter defaults to *SBSD. If you use this default, but do not have a class with that name, the prestart jobs may fail. You should specify the same class as you would use for interactive jobs, that is, the class defined with the CRTCLS command when you created the subsystem.
2. The STRJOBS(*NO) parameter must be specified because the prestarted jobs are started by the CICS control region during initialization.
3. WAIT(*YES) must always be specified otherwise your requests may be lost from the system before CICS is ready to start them.
4. The MAXJOBS parameter of the ADDPJE command should be equal to the total number of receive sessions allocated for the control region in the TCS entries (see "Defining remote CICS systems, ADDCICSTCS" on page 39).

The ADDPJE command screen is shown in Figure 8.

```
                      Add Prestart Job Entry (ADDPJE)

Type choices, press Enter.

Subsystem description  . . . . .               Name
  Library  . . . . . . . . . . .     *LIBL      Name, *LIBL, *CURLIB
Program  . . . . . . . . . . . .               Name
  Library  . . . . . . . . . . .     *LIBL      Name, *LIBL, *CURLIB
User profile . . . . . . . . . .   QUSER        Name
Start jobs . . . . . . . . . . .   *YES         *YES, *NO
Initial number of jobs . . . . .   3            1-1000
Threshold  . . . . . . . . . . .   2            1-1000
Additional number of jobs  . . .   2            0-999
Maximum number of jobs . . . . .   *NOMAX       1-1000, *NOMAX









                                                                      Bottom
F3=Exit  F4=Prompt  F5=Refresh  F12=Cancel  F13=How to use this display
F24=More keys
Parameter SBSD required.                                               +
```

Figure 8. The ADDPJE command screen

The parameters on the ADDPJE command screen are as follows:

Subsystem description (SBSD)
Specifies the name and library of the subsystem description to which the prestart job entry is being added. If no library qualifier is given, *LIBL is used to find the subsystem description.

The possible library values are:

***LIBL**
The library list is used to locate the subsystem description.

***CURLIB**
The current library for the job is used to locate the subsystem description. If no library is specified as the current library for the job, QGPL is used.

library-name
The library where the subsystem description is located.

Program (PGM)
Specifies the name and library of the program run by the prestart job. This program name is used to match an incoming program start request with an available prestart job. If the program does not exist when the entry is added, a library qualifier must be specified because the qualified name is kept in the subsystem description.

Note: Two entries with the same program name can exist in a single subsystem description, but they must have different library names.

The possible library values are:

***LIBL**
The library list is used to locate the program.

***CURLIB**
The current library for the job is used to locate the program. If no library is specified as the current library for the job, QGPL is used.

library-name
The library in which the program is located.

User profile (USER)
Specifies the user profile under which the prestart job runs.

Note: If a user profile is sent on a program START request, OS/400 checks the password, user profile, and its authority to access the communications device, library, and program. However, none of the attributes of the program START request user profile are given to the prestart job. If a user profile is not sent on a procedure START request, the user profile from the communications entry is used with the same rules applied. The Change Prestart Job (CHGPJ) command can be used to adopt the attributes of the user profile or job description associated with the program START request.

The possible values are:

QUSER
The supplied QUSER user profile is used.

user-profile-name
The name of the user profile used for the prestart job.

Start jobs (STRJOBS)
Specifies whether the prestart jobs are started at the time the subsystem is started.

Note: Changing this value when the subsystem is active has no effect until the subsystem is ended and started again.

The possible values are:

***YES**
The prestart jobs are started at the time the subsystem is started.

***NO**
The prestart jobs are not started at the time the subsystem is started. The Start Prestart Jobs (STRPJ) command must be used to start these prestart jobs.

Initial number of jobs (INLJOBS)
Specifies the initial number of prestart jobs that are started when the subsystem named in the subsystem description (SBSD) parameter is started.

Notes:

1. The value of this parameter must be less than or equal to the value in the maximum number of jobs (MAXJOBS) parameter.
2. The value of this parameter must be greater than or equal to the value in the threshold (THRESHOLD) parameter.

The possible values are:

3 Three prestart jobs are started when the subsystem is started.

initial-active-jobs
The number of prestart jobs that are started when the subsystem is started. Valid values are in the range 1 through 1000.

Threshold (THRESHOLD)
Specifies a minimum number of available jobs, used to determine when additional prestart jobs are started. When the pool of available jobs (jobs available to service a program START request) falls below this number, more jobs, as specified on the additional number of jobs (ADLJOBS) parameter, are started and added to the available pool.

Note: The value in this option must be less than or equal to the value specified in the initial number of jobs (INLJOBS) parameter.

The possible values are:

2 When only one prestart job is available, the number of jobs specified on the additional number of jobs (ADLJOBS) parameter is started.

threshold-value
The minimum number of prestart jobs that must be available before additional prestart jobs are started. Valid values range from 1 through 1000.

Additional number of jobs (ADLJOBS)
Specifies the additional number of prestart jobs that are started when the number of prestart jobs falls below the value specified in the threshold (THRESHOLD) parameter.

Note: The value specified on this option must be less than the value specified in the maximum number of jobs (MAXJOBS) parameter.

The possible values are:

2 Two additional prestart jobs are started.

additional-active-jobs
Specify the number of additional prestart jobs to start. Valid values are in the range 0 through 1000.

Maximum number of jobs (MAXJOBS)
Specifies the maximum number of prestart jobs that can be active at the same time for this prestart job entry. This value includes prestart jobs that are servicing or waiting to service a procedure START request, and prestart jobs that are being started as a result of reaching the value specified on the threshold (THRESHOLD) parameter.

Notes:

1. The value of this parameter must be greater than or equal to the value specified on the initial number of jobs (INLJOBS) parameter.
2. The value of this parameter must be greater than the value specified on the additional number of jobs (ADLJOBS) parameter.

The possible values are:

***NOMAX**
There is no limit on the number of prestart jobs that can be active at the same time.

maximum-jobs
Specify the maximum number of prestart jobs that can be active at the same time. Valid values range from 1 through 1000.

Adding configuration list entries

To allow OS/400 APPN to identify your CICS system as a local LU, you need to register the CICS/400 APPLID in the APPN local location list. You do this using the add configuration list entries (ADDCFGLE) command and specifying TYPE(*APPNLOCL). The ADDCFGLE command screen is shown in Figure 9.

```
                  Add Configuration List Entries (ADDCFGLE)

Type choices, press Enter.

Configuration List Type  . . . .>  *APPNLCL     *APPNLCL, *APPNRMT...
APPN Local Location Entry  . . ._
  Local Location Name  . . . . .   ________      Name
  Entry 'Description'. . . . . .   *BLANK_____________________________
               + For More Values_

                                                                  Bottom
F3=Exit  F4=Prompt  F5=Refresh  F12=Cancel  F13=How to use this display
F24=More keys
```

Figure 9. The ADDCFGLE command screen

Similarly, to identify the remote CICS/400 system to APPN, you need to register it in the APPN remote location list, using the ADDCFGLE command and specifying TYPE(*APPNRMT).

You must tie up the local CICS system with the remote CICS system in the same way as the APPC device description.

You can also add configuration list entries using the WRKCFGL command and following the panels.

Working with the configuration

You can dynamically control your configuration with the Work with Configuration Status (WRKCFGSTS) command, which allows you to display and work with configuration status functions. The WRKCFGSTS display shows status information for network interfaces, lines, controllers, devices, and jobs associated with devices. The display can be for a location, or for one or more network interfaces, lines, controllers, or devices. All attached configuration descriptions are shown for each network interface, line, controller, or device description selected.

Options available on the Work with Configuration Status display are to vary status, end or resume recovery, hold or release a device, work with a job, or display location path.

Typical uses of WRKCFGSTS include:

- As a first action when a communication problem occurs, to check the status of lines, controllers and devices
- To vary off and on controllers and devices
- To check that a job has attached to a communications device.

For guidance in the use of the WRKCFGSTS command, see *Communications Management*, SC41-5406-02. Reference information for the WRKCFGSTS command is in the Control Language (CL) topic in the iSeries Information Center.

Setting up the CICS resource definitions

A terminal control system (TCS) entry must be defined for each remote CICS system with which CICS/400 communicates. You do this using the ADDCICSTCS command. You also need to use the ADDCICSSIT command to define the network id on the system initialization table.

Defining remote CICS systems, ADDCICSTCS

The ADDCICSTCS command adds an entry to the CICS/400 terminal control system table (TCS). Each entry defines a connection between the local CICS/400 system and a remote system. For full details, see the *CICS for iSeries Administration and Operations Guide*.

TCS entries may also be installed dynamically, using either the INSCICSGRP CL command or the CICS-supplied resource definition transaction CEDA. See the *CICS for iSeries Administration and Operations Guide* for details.

The TCS table is the bridge at the CICS level between the local CICS/400 system and remote CICS systems. (The TCS also exists on CICS OS/2. CICS/mainframe users use CONNECTION and SESSION definitions, (or TERMINAL and TYPETERM definitions for single-session links) to define remote CICS systems.)

The following is guidance information on selected parameters of the ADDCICSTCS command.

CICS system (SYSID)
Specifies the name of the connection being defined. This name is used in the SYSID parameter of CL commands that define remote resources (such as ADDCICSPCT or ADDCICSFCT), and in the SYSID parameter of CICS/400 commands that can specify remote systems (such as EXEC CICS WRITEQ TS or EXEC CICS START).

Network (NETWORK)
Specifies the network name used to identify the remote CICS system. This name must be the same as the RMTLOCNAME parameter of the CRTDEVAPPC command that defines the OS/400 communications device, and as the APPLID in the system initialization table (SIT) of the remote CICS system.

Mode (MODE)
Specifies the mode group used for intercommunication with the remote system. For a list of available mode groups, issue the DSPMODSTS command for the OS/400 device used by the connection[5] . The group must be the same as that in the MODE parameter of the CRTDEVAPPC command that defines the OS/400 communications device.

Code page (CDEPAGE)
Do not specify this parameter. Allow it to default to **CDEPAGE(*SYSVAL)**.

CICS system status (SYSSTS)
Specifies the status of the connection, which can be *ENABLED or *DISABLED. For communication to occur, the OS/400 device associated with the connection must be varied on.

Outbound session prefix (SNDPFX)
Specifies a prefix to be concatenated to the outbound session number. This prefix must be unique in the local CICS/400 system. For a single-session connection, this prefix must be the same as the inbound session prefix. If no outbound sessions are required, specify **SNDPFX(*NONE)**.

Inbound session prefix (RCVPFX)
Specifies a prefix to be concatenated to the inbound session number. This prefix must be unique in the local CICS/400 system. For a single-session connection, this prefix must be the same as the outbound session prefix. If no inbound sessions are required, specify **RCVPFX(*NONE)**.

Indirect CICS system (INDSYS)
Specifies an intermediate CICS system that is used by the connection to relay communication with the target system. This parameter is used solely to support transaction routing. If the connection is direct, specify **INDSYS(*NONE)**.

Figure 10 on page 41 illustrates the need for an indirect system connection. TERM, a terminal attached to the CICS/400 system TOR, wants to transaction route to a transaction in AOR2. TOR and AOR2 are not directly connected, but AOR2 has a connection CONA to AOR1, and AOR1 has a connection CONT to TOR. Transaction routing is possible if the remote transaction is defined in TOR with SYSID pointing to AOR1, and in AOR1 with SYSID pointing to AOR2 (see “Serial connections” on page 74). AOR1 relays the transaction routing request to AOR2 but the accompanying shipped terminal definition specifies SYSID(CONT). To be able to communicate with its initiating terminal, the transaction requires that AOR2 has a TCS entry for TOR with SYSID and INDSYS specified as below:

```
ADDCICSTCS LIB(CICSWORK) GROUP(GROUP) SYSID(CONT) INDSYS(CONA)
```

5. Do not use SNASVCMG and CPSVCMG, which are system mode groups that cannot be used for CICS intercommunication.

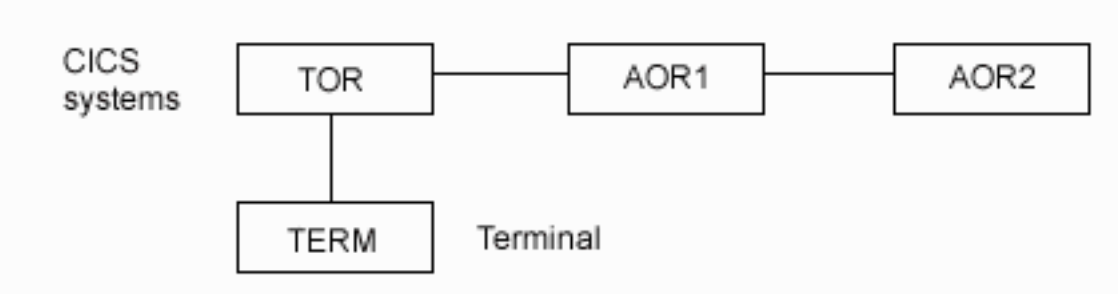


Figure 10. Indirectly connected CICS systems

Send limit (SNDLMT)
Specifies the number of outbound sessions available on the connection. Numbers in the range 00 through the specified limit are concatenated to the SNDPFX to create a range of unique session identifiers. For single-session connections, specify **SNDLMT(1)**.

Receive limit (RCVLMT)
Specifies the number of inbound sessions available on the connection. Numbers in the range 00 through the specified limit are concatenated to the SNDPFX to create a range of unique session identifiers.

A task started by an attach sent to a CICS/400 system abends if no inbound session is available. To avoid this possibility, set the RCVLMT parameter to the total number of conversations available on the device.

The definitions of conversations in mainframe CICS and CICS/400 differ. With CICS/400, a conversation is a temporary connection of a program to a session, and can be either synchronous (both the source and the target programs are communicating, as with mainframe CICS), or asynchronous (the source program has completed and detached from the session, leaving the session available for work, but the target system is still attached and has access to all the data sent by the source program).

It is recommended that the MAXCNV parameter on the mode definition is set to a higher value than MAXSSN. For more information about defining sessions and connections, see the *Communications Configuration* book. For single-session connections, specify **RCVLMT(0)**.

Remote network indicator (RMTNETID)
Specifies the network indicator of the remote system. If the remote system is an OS/400 subsystem (as CICS/400 is), the network indicator is the **Local network ID**[6] of the remote OS/400 system. This value must be the same as the RMTNETID parameter of the CRTDEVAPPC command that defines the OS/400 device to be used.

Example

This command adds a TCS entry, called CONR, to a CICS/400 group, called GROUP, in an OS/400 library called CICSWORK.

```
ADDCICSTCS LIB(CICSWORK) GROUP(GROUP) SYSID(CONR) NETWORK(REMTCICS)
           SNDPFX(SP) SNDLMT(5) RCVPFX(RP) RCVLMT(5) RMTNETID(REMTNET)
           MODE(MODEGRP)
```

The SYSID parameter is the name of the connection, and is the local name of the remote system defined by this TCS entry. The system places no constraints on the

6. To display the local network ID of an OS/400 system, enter the DSPNETA command on the system whose ID is required.

SYSID value. For a remote CICS/400 system, it is often convenient to use its OS/400 subsystem name (REMT in Figure 1 on page 12).

The SYSID value in this command is used in remote resource definitions to identify the resource-owning region, and in DPL and EXEC CICS START commands to identify the target region. For guidance on defining resources that are owned by remote CICS systems, see Chapter 4 to Chapter 7.

The NETWORK, RMTNETID, and MODE parameters (together with the APPLID parameter in the system initialization table) point to a device description (CRTDEVAPPC command). The device description defines the remote system and points to the local OS/400 controller and line that control the link. See "The APPC connection" on page 43.

CICS/400 system definition table, ADDCICSSIT

The system initialization table (SIT) contains information used to start and control the CICS/400 control region. There is only one system initialization parameter related to intercommunication. Use the ADDCICSSIT command to create an entry for this parameter in the CICS/400 SIT. For full details of the ADDCICSSIT command, see the *CICS for iSeries Administration and Operations Guide*.

Application network name (APPLID)
The name by which the CICS control region is known in the network. This is equivalent to an APPLID in CICS for OS/390.

The local SYSID is taken from the name of the control region, as specified on the STRCICS command.

Working with the configuration

You can dynamically control your configuration with the Work with Configuration Status (WRKCFGSTS) command, which allows you to display and work with configuration status functions. The WRKCFGSTS display shows status information for network interfaces, lines, controllers, devices, and jobs associated with devices. The display can be for a location, or for one or more network interfaces, lines, controllers, or devices. All attached configuration descriptions are shown for each network interface, line, controller, or device description selected.

Options available on the Work with Configuration Status display are to vary status, end or resume recovery, hold or release a device, work with a job, or display location path.

Typical uses of WRKCFGSTS include:

- As a first action when a communication problem occurs, to check the status of lines, controllers and devices
- To vary off and on controllers and devices
- To check that a job has attached to a communications device

For guidance in the use of the WRKCFGSTS coommand, see the *Communications Management* book. Reference information for the WRKCFGSTS command is in the Control Language (CL) topic in the iSeries Information Center.

Summary

This section summarizes CICS/400 communication configuration described in this chapter.

How the subsystem is associated with CICS resource definitions and with OS/400

Subsystem LOCL is initiated by the command STRSBS LOCL. The CICS control region is then initiated by the command STRCICS LOCL, which allows the operator to enter parameters that specify the OS/400 library and CICS/400 group to be used. The CICS/400 group contains a single system initialization group (SIT). In the SIT, the GLT parameter specifies the group list table that contains the resource definitions used by this CICS system. For intercommunication, the central CICS resource definitions are the terminal control system (TCS) entries that define remote CICS systems to CICS/400. TCS entries are described under "Defining remote CICS systems, ADDCICSTCS" on page 39.

The APPC connection

CICS intersystem communication uses an APPC device defined to OS/400 by a CRTDEVAPPC command. A CICS subsystem requires a device that matches parameters in the ADDCICSTCS and ADDCICSSIT commands as shown in Table 3.

Table 3. Matching CRTDEVAPPC to ADDCICSTCS and ADDCICSSIT

CRTDEVAPPC	ADDCICSTCS	ADDCICSSIT
LCLLOCNAME		APPL
RMTLOCNAME	NETWORK	
RMTNETID	RMTNETID	
MODE	MODE	

In Figure 1 on page 12, flags 3, 4, 5, and 6 indicate these matching fields. The CRTDEVAPPC command points on to controller and line definitions that complete an OS/400 configuration to support CICS/400.

Figure 1 also shows the fields that must match in the remote and local device definitions (CRTDEVAPPC). Each LCLLOCNAME parameter must match the remote RMTLOCNAME parameter (in CICS terms, this means that the APPL parameter in each ADDCICSSIT command must match the NETWORK parameter in the remote ADDCICSTCS command). The mode parameters in each CRTDEVAPPC command must have the same value. This means that the mode definitions at each end have the same name, but ***not*** that they necessarily have the same content.

When CICS/400 communicates with a CICS product other than another CICS/400 system:

- RMTLOCNAME is the application identifier of the remote system (in a CICS/mainframe system, the APPLID parameter in the SIT).
- LCLLOCNAME matches a field in the remote system's definition of the connection (in a CICS/mainframe system, the NETNAME[7] parameter in the CONNECTION definition).

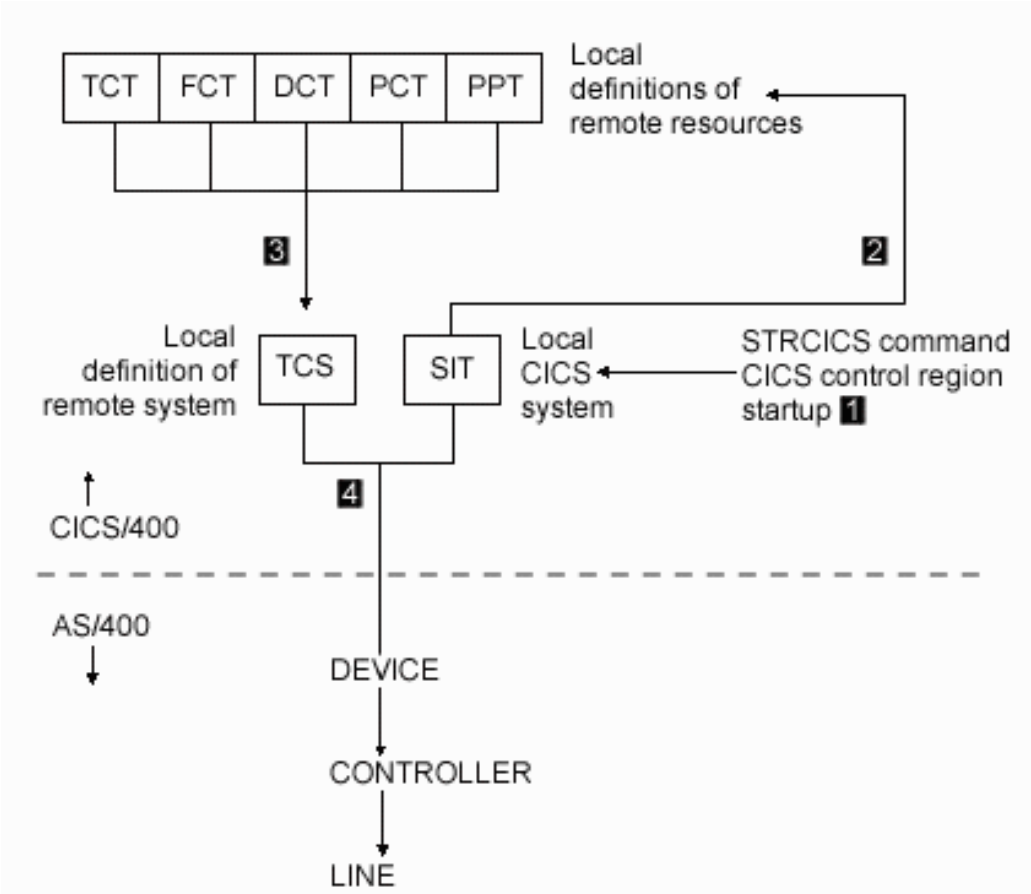
7. NETNAME defaults to the name of the CONNECTION, which in this example is CICS/mainframe's name for the CICS/400 system.

- The MODE name must match the name of the mode definition associated with the remote system's session (in a CICS/mainframe system, the MODENAME parameter in the SESSIONS definition).

For more information about the definition of devices, controllers, and lines, refer to the *Communications Configuration* book.

How resource definitions are connected

Figure 11 on page 45 shows how the different resource definitions are connected, including the link between CICS/400 and OS/400.



1 The STRCICS command enables the operator to specify the CICS group, which contains the system initialization table (SIT).
2 In the SIT, the GLT parameter specifies the group list table that contains all CICS resource definitions to be used.
3 All *remote* resource definitions point to the TCS entry that defines the remote system.
4 The TCS and SIT jointly specify the AS/400 device.

Key to abbreviations

TCT Terminal control table
DCT Destination control table
PPT Processing program table
SIT System initialization table
FCT File control table
PCT Program control table
TCS Terminal control system table

Figure 11. How resource definitions are connected

Intrasystem communication

This section highlights some of the different parameter values that are required to set up intrasystem communication (intercommunication between two subsystems on the same OS/400 system).

Line definition

As Figure 1 on page 12 indicates, an OS/400 line definition is required for CICS/400 intersystem communication. For intrasystem communication no line definition is necessary.

Controller definition

The controllers must be defined with a CRTCTLAPPC command that specifies **LINKTYPE(*LOCAL)**.

```
CRTCTLAPPC CTLD(STLCLCTL01) LINKTYPE(*LOCAL) ONLINE(*YES) CMNRCYLMT(2 5)
           TEXT('Local controller for intrasystem communication')
```

Device definition

As each APPC device is unidirectional, two devices must be defined for full-duplex communications between two subsystems. Also, both devices must use the same *LOCAL controller.

You must specify APPN(*NO) on the CRTDEVAPPC command.

Example of intrasystem communication definitions

The following definitions connect 2 CICS/400 regions, ISC1 and ISC2, for intrasystem communications.

Step 1

Define the local intrasystem controller for CICS/400 communications.

```
CRTCTLAPPC CTLD(&CTL) LINKTYPE(*LOCAL) ONLINE(*YES) +
           TEXT('LOCAL CONTROLLER FOR INTRASYSTEM COMMS')
```

The parameters used are:

CTLD(&CTL)
Specifies the name of the controller description.

LINKTYPE(*LOCAL)
Specifies that this controller is attached to a local line.

ONLINE(*YES)
Specifies that this controller is to be varied on automatically at initial program load.

TEXT('LOCAL CONTROLLER FOR INTRASYSTEM COMMS')
Describes the controller, for documentation purposes.

Step 2

Create a mode group.

```
CRTMODD    MODD(CICSISC0) MAXSSN(20) MAXCNV(20)
```

The parameters used are:

MODD(CICSISC0)
Specifies the name of this mode description.

MAXSSN(20)
Specifies that 20 is the maximum number of active sessions that can be established for this mode.

MAXCNV(20)
Specifies that 20 is the maximum number of conversations that can be established at the same time with the other system.

This mode definition uses the default class-of-service #CONNECT.

Step 3

Define the local intrasystem APPC devices, one in each direction.

```
CRTDEVAPPC DEVD(ISC1ISC2) RMTLOCNAME(ISC2APPL) +
           LCLLOCNAME(ISC1APPL) CTL(&CTL) +
           MODE(CICSISC0) APPN(*NO)

CRTDEVAPPC DEVD(ISC2ISC1) RMTLOCNAME(ISC1APPL) +
           LCLLOCNAME(ISC2APPL) CTL(&CTL) +
           MODE(CICSISC0) APPN(*NO)
```

The parameters used are:

DEVD(ISC1ISC2)/(ISC2ISC1)
Specifies the name of the device description.

RMTLOCNAME(ISC2APPL)/(ISC1APPL)
Specifies the unique location name that identifies the other CICS/400 region.

LCLLOCNAME(ISC1APPL)/(ISC2APPL)
Specifies the unique location name that identifies the local CICS/400 region.

CTL(&CTL)
Specifies the name of the controller to which this device is attached. This is the same as specified in the CTLD parameter of the CRTCTLAPPC command.

MODE(CICSISC0)
Specifies the name of the mode that define the sessions characteristics. This is the same as specified in the MODD parameter of the CRTMODD command.

APPN(*NO)
Specifies that this device is not for Advanced Peer-to-Peer Networking.

Step 4

Add 2 Terminal Control System (TCS) entries, one for each CICS/400 region, to allow the 2 regions to communicate with each other.

In ISC1:

```
ADDCICSTCS LIB(ISC1LIB) GROUP(ISC1) SYSID(ISC2) +
           NETWORK(ISC2APPL) MODE(CICSISC0) +
           SNDPFX(SA) SNDLMT(5) RCVPFX(RA) RCVLMT(5)
```

In ISC2:

```
ADDCICSTCS LIB(ISC2LIB) GROUP(ISC2) SYSID(ISC1) +
           NETWORK(ISC1APPL) MODE(CICSISC0) +
           SNDPFX(ST) SNDLMT(5) RCVPFX(RT) RCVLMT(5)
```

The parameters used are:

LIB(ISC1LIB)/(ISC2LIB)
Specifies the OS/400 library containing the CICS/400 resource definition group.

ADDCICSSIT

GROUP(ISC1)/(ISC2)
Specifies the CICS/400 resource definition group name.

SYSID(ISC2)/(ISC1)
Specifies the name of this TCS entry.

NETWORK(ISC2APPL)/(ISC1APPL)
Specifies the network name used to identify the other CICS system. This is the same as specified in the RMTLOCNAME parameter of the CRTDEVAPPC command.

MODE(CICSISC0)
Specifies the mode group used for intercommunication with the remote system. This is the same as specified in the MODD parameter of the CRTMODD command.

SNDPFX(SA)/(ST)
Specifies a prefix to be used to create unique outbound session identifiers.

SNDLMT(5)
Specifies that 5 is the maximum number of outbound sessions available on the connection.

RCVPFX(RA)/(RT)
Specifies a prefix to be used to create unique inbound session identifiers.

RCVLMT(5)
Specifies that 5 is the maximum number of inbound sessions available on the connection.

Chapter 3. CICS/400 server support for the CICS client family

The CICS client family allows workstations to access CICS resources in a remote CICS server such as CICS/400, CICS for OS/390, CICS for Open Systems, or CICS for OS/2, but place only moderate demands on the workstation itself. When connected to a server, a client has access not only to the resources local to that server, but also to all the remote resources to which the server has access. The server can act as the client's gateway into an entire CICS network.

Overview

CICS Client to CICS/400 communications may use either APPC or TCP/IP connectivity. CICS clients connect using synchronization level 1. Clients are available for:

- AIX®
- Microsoft® Windows®
- OS/2
- Solaris
- HP-UX
- Linux 390

See the *CICS Clients Administration*, SC33-1436 book or *CICS Family: Client Server Programming*, SC33-1435 book for details.

What the CICS client does

CICS client applications are written to programming interfaces provided on the client workstation:

- External Presentation Interface (EPI)
- External Call Interface (ECI)

The EPI allows existing CICS applications to exploit workstation interfaces such as user-friendly Graphical User Interfaces (GUIs) on CICS clients, without the need for changes to the CICS application.

The ECI allows the design of new client/server applications. Typically the business logic is kept on a CICS server, and the presentation logic is implemented on the workstation and takes full advantage of the GUI.

CICS clients also provide a CICS 3270-terminal emulator that enables existing 3270 CICS transactions to be displayed on the client as they would be on a real 3270 device.

The client support on the workstation converts the EPI and ECI calls into standard CICS ISC flows.

What the CICS/400 server does

A CICS client appears to its server much like a full-function CICS system sending CICS intersystem communication requests using the EPI and ECI interfaces.

However, unlike full CICS systems, clients may not initially be known to the CICS/400 server or OS/400. The following facilities can be enabled with minimal setup, and permit previously unknown clients to be added to a network and to have immediate access to CICS/400, without requiring the user to create individual definitions in either CICS/400 or OS/400.

The OS/400 automatic configuration facility
can be used to create controller and device descriptions for new clients as they appear. For more information on this facility, see "Automatic configuration of dynamic devices" on page 57.

OS/400 routing-table entries
ensure that client requests are processed by CICS/400. These can be set up once and are used by all clients. For a discussion of the routing entries that are required, see "Required routing entries" on page 54.

CCIN and CTIN
are two CICS server transactions sent by the client to the server by way of introduction. The CCIN transaction installs a TCS entry for the client and the CTIN transaction installs a remote terminal entry where necessary. For more information regarding CCIN and CTIN, see "Resource definition".

Resource definition

The resource definition requirements for CICS client support are explained below:

- The supplied CICS resource definition group AEGCLI must be installed to enable the use of transactions CCIN and CTIN.
- To enable usage of the ECI, EPI, and terminal emulator application interfaces at the client, a system entry must be installed in the CICS/400 control region for each client. Installation is handled automatically by the CCIN transaction (CICS Client INstall), which installs a system entry at client startup, and uninstalls it at client shutdown.
- A remote terminal entry must be installed in the CICS/400 control region for each client that uses the EPI interface or the 3270 emulator. This is handled automatically by the CTIN transaction (CICS Terminal INstall), which installs and uninstalls remote terminal entries as requested by the client application.
- Data conversion definitions may need to be created in the CVT for CICS/400 programs that are called by client ECI applications. Use the ADDCICSCVT command to create CVT entries.
- All clients must have a valid OS/400 user ID and password, and must be authorized to all CICS/400 resources to which they require access. Administration of user IDs and passwords is done using OS/400 commands.

Although the CCIN and CTIN transactions provide automatic setup facilities you may choose to define client TCS or TCT entries manually using the ADDCICSTCS or ADDCICSTCT commands. When a CCIN or CTIN install request detects an installed definition, it is used provided it is not currently in use. When an uninstall request is received, a manually-defined entry is not deleted, but remains available for future use.

Client system entry

CCIN is a CICS/400 internal transaction that is invoked at client startup and shutdown. When the client is started a CCIN system install request is passed to the

server requesting install of a system entry for this client. When the client is shut down a CCIN system uninstall request is passed to the server, requesting removal of the associated system entry.

CCIN install
The install transaction generates a system entry for a client as follows:

- When no entry exists a new entry is created for the client.
- When an entry already exists and is not currently in use, all dependent sessions and autoinstalled remote terminals are deleted, and the entry is used for the client.

CCIN uninstall
CCIN uninstalls only those entries that it has installed itself.

- When an entry exists it is checked to determine if it was installed using CCIN. If so, it is deleted, otherwise it must have been installed from an ADDCICSTCS definition, and is not deleted, but left for future use.
- When no entry exists, an error message is put on the job and history logs.

Default settings
A new client system entry uses the following default values:

CICS system (SYSID)
Sequentially allocated in the range "**-AAA**" through "**-ZZZ**".

Network (NETWORK)
Set from the OS/400 device description. Although the CICS client may pass a network name in the CCIN install request, this name is not used to set this field. As a default, the client does not pass the network name.

Mode (MODE)
Set from the OS/400 device description.

Code page (CDEPAGE)
Set from the **CODEPAGE** parameter sent by the client. When the code page is not recognized on the OS/400, a default may be installed as described in "Data conversion" on page 53.

Status (SYSSTS)
Set to ***ENABLED**

Outbound session prefix (SNDPFX)
Set to a 2-character value. The characters are binary and not displayable, and exclude blanks, nulls, alphanumerics, and the 3 special characters $[8] , @, and #.

Available outbound sessions (SNDLMT)
Set to **1**.

Inbound session prefix (RCVPFX)
Set to a 2-character value. The first character is the second character of the send prefix. The second character is the first character of the send prefix.

Available inbound sessions (RCVLMT)
Set to **10**.

Indirect CICS system (INDSYS)
Set to ***NONE**.

8. National currency symbol.

Remote network indicator (RMTNETID)
Set from the OS/400 device description.

When an existing definition is used by a client, the only fields that take the default value are Code page (CDEPAGE) and Status (SYSSTS). The other fields remain unchanged.

Client terminal entry

CTIN is a CICS/400 internal transaction that is invoked when the client requests a terminal, for example, through use of the client terminal emulator or from an EPI application. A request is passed to the server to install a remote terminal entry for this client. When the application ends, a CTIN terminal uninstall request is passed to the server, requesting removal of the specified terminal entry.

CTIN install
The transaction uses the parameters from the client to create a remote terminal entry. The following sequence is used:

- When a **netname** is passed, the transaction looks for the terminal entry with that name. When a valid remote terminal entry is found it is used.
- When a **modelid** is passed the transaction looks for the model terminal entry with that name. When a valid model entry is found CTIN creates a new remote terminal entry using the values from the model.
- When neither **netname** nor **modelid** is supplied, the transaction builds a new remote terminal entry using CICS/400 internal values.

When the install is successful, terminal details are returned to the client. When the install fails, for example, when an unknown netname or modelid was specified, an installation-failed indicator is returned to the client and error messages are issued to the job and history logs.

CTIN uninstall
CTIN uninstalls only those entries that it has installed itself. The transaction uses the terminal identifier passed from the client to locate the remote-terminal entry, as follows:

- The transaction locates the remote-terminal entry, using the termid parameter, and deletes it if it is a valid autoinstalled client terminal.

No messages are returned to the client for uninstall requests. When the transaction fails to uninstall the terminal error messages are issued to the job and history logs.

Default settings
When a netname (terminal id) is passed from the client, default settings are not required because the parameters in the terminal definition are used. Defaults are required only when the netname parameter is not passed. Table 4 describes the default terminal parameters and how they override parameters in a model entry. If you wish to define a model terminal for use by clients you must use the ADDCICSTCT command. You may choose to base the definition on an existing surrogate terminal.

Table 4. Default terminal control table parameters

Parameter name	Default value	Override model?
CICSDEV	Generated automatically within the range ″?AAA″ through ″?ZZZ″	Yes

Table 4. Default terminal control table parameters (continued)

Parameter name	Default value	Override model?
SYSID	System identifier for the client	Yes
RMTDEV	As for CICSDEV	If not defined
DEVTYPE	3270	No
PRTFILE	Blank	No
DEVD	Blanks	No
DEVMODEL	*TERMINAL	Yes
NETWORK	SNA LU name	Yes
ALTSUFFIX	*NONE	No
DEVSTS	*ENABLED	Yes
ATISTS	*YES	No
TTISTS	*YES	No
USRARASIZE	0	No
DEVCHRID	From the CTIN code page parameter. If no parameter is provided it is copied from the client system's entry	Yes
TRANSID	*ANY	No
KATAKANA	*NO	No
SOSI	*NO	No
UNATTEND	*NO	No
UCTRAN	*YES	No
DSCOMP	*NO	No
ALTSCN	*NONE	No
VALIDATION	*NO	No
LIGHTPEN	*NO	No
SHIP	*YES	No
DEVACQ	*NO	No

Data conversion

ECI applications

When CICS client ECI applications send or receive ASCII data, they expect the CICS/400 server to handle any ASCII-EBCDIC conversion required. When CICS/400 receives the inbound transaction, it scans user-defined conversion table (CVT) entries to determine whether this CICS program requires data conversion, and to establish which OS/400 conversion table should be used. Conversion table entries are created using the ADDCICSCVT command.

When a CICS program is used by clients with different ASCII code pages, the code page in the CVT is temporarily overridden by the value installed in the client system entry. The transaction uses the client code page while leaving the CVT entry unchanged. In this way, many different client code pages can be supported. If the client code page is not recognized by CICS/400, the system entry defaults as follows:

- When the client parameters indicate an ASCII workstation the entry is set to **850**, the multi-lingual ASCII code page.
- When an EBCDIC client is indicated the entry is set to the OS/400 system code page.

Note: CVT routines are dependent on the required code page being defined to OS/400. You must ensure that table entries exist in QSYS and QUSRSYS for any code page that a client wishes to use.

EPI applications and terminal emulators

CICS client EPI applications transaction-route ASCII data to CICS/400, where any ASCII-EBCDIC conversions must be handled. The code-page parameter in the CTIN install is used to indicate the client code page for EPI, and is installed in the remote-terminal entry.

No additional setup is required to convert EPI data streams, as the value in the remote-terminal entry is used for inbound and outbound data conversion. When the remote-terminal entry is created from a model terminal definition, the code page is set using the following priority:

- If a code page is provided in the CTIN install it is used.
- If no code page is provided then the value in the system entry is used. This can be the default ASCII multi-lingual code page **850**.

Restrictions

- A system restart causes the deletion of all system and terminal definitions installed by the CCIN and CTIN transactions. The deletion of these definitions results in the failure of any outstanding ATI requests for client terminals.
- Programs to be run using a CICS client terminal emulator session must use the CICS API for terminal I/O. Transactions such as CEDA, which use the OS/400 interface, cannot be run from a client terminal.

Required routing entries

The routing table is the part of a subsystem that identifies the program to run when the subsystem receives work. The number of routing entries required depends on the number of CICS/400 control regions associated with the subsystem and on whether the subsystem handles non-CICS requests. The different situations are summarized in Figure 12 on page 56 and discussed below.

In a subsystem dedicated to a specific CICS/400 control region, a single routing-table entry can ensure that all communications requests cause the program AEGISICC to be started. This CICS/400 program can handle all types of CICS ISC flows, but only for a single, specific control region. This is case **1** in Figure 12.

If a subsystem is *not* dedicated to a single CICS/400 system, CICS/400 uses a special program AEGISRTR that can decide at run time which CICS/400 control region should be given the inbound request. In a subsystem that handles *only* CICS/400 work, but possibly for a number of control regions, replacing AEGISICC with AEGISRTR in the appropriate routing-table entry is all you need to do. This is case **2** in Figure 12.

In the most common case, the communications entries are such that the clients' devices are associated with QBASE or QCMN, which handle more than just CICS/400 requests. Here, several routing-table entries are needed to ensure that

CICS requests cause AEGISRTR to be evoked, but that non-CICS requests are handled as normal (and are *not* directed to AEGISRTR). This is case **3** in Figure 12.

Routing entries in default subsystem

If clients' devices are associated with QBASE or QCMN (case **3** in Figure 12 on page 56), routing-table entries for the following transactions need to be defined as follows:

CCIN A transaction that installs a CICS client system (TCS) definition.

CPMI The standard CICS function-shipping transaction (for ASCII devices). CPMI is used to flow the DPL requests generated as a result of ECI calls.

CTIN A transaction that installs a CICS client terminal (TCT) definition.

CRTE The standard CICS transaction-routing transaction. CRTE is used to flow requests generated as a result of EPI calls.

The routing entries for CCIN, CPMI, CTIN, and CRTE must have a lower sequence number than any routing-table entry specifying comparison data of "PGMEVOKE" starting in column 29 of the routing data.

Note: When Client Access/400 for DOS is used to connect clients to CICS/400, special consideration must be given to the routing entries. Specifically, the routing entry for mode QPCSUPP must have a higher sequence number than CICS routing entries, to ensure that CICS requests are properly routed.

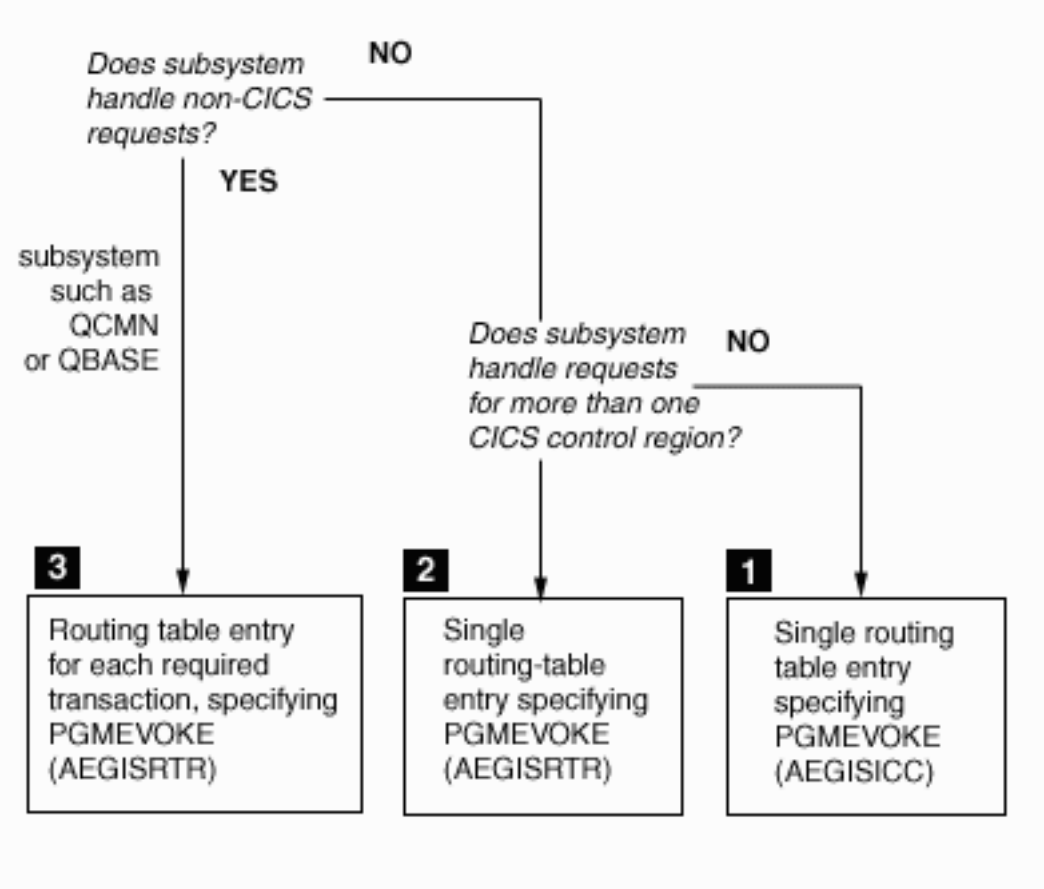


Figure 12. Routing entries required by CICS/400 server

Figure 13 on page 57 shows a sample ADDRTGE command screen for the CCIN routing definition. The fields of special interest are discussed below. The other values are installation-dependent and are explained in "Adding routing entries" on page 28, which also gives the definitive descriptions of the fields mentioned here.

Subsystem description (SBSD)
Specifies the subsystem that handles incoming requests from clients. Typically, this is QCMN or QBASE.

Routing entry sequence number (SEQNBR)
Specifies the routing sequence number. Sequence numbers determine the order in which routing-entry compare values are tested against the routing data in an incoming request. Ensure that the entries you create have a lower sequence number than any routing table entry specifying comparison data of "PGMEVOKE" starting in column 29 of the routing data.

Comparison data (CMPVAL)
Specifies the routing data used to identify incoming client requests. For an inbound communication request, the literal text "PGMEVOKE" starts in position 29 in the routing data, and the transaction program name starts in position 37. You need to add entries for "CCIN", "CTIN", "CPMI", and "CRTE", starting in position 37.

If the subsystem to which you are adding this entry does not receive non-CICS communication requests, it is sufficient to have a single RTG entry specifying "PGMEVOKE" starting in position 29.

Program to call (PGM)
Specifies the special CICS/400 routing program AEGISRTR, or AEGISICC if special case **1** in Figure 12 applies.

```
                          Add Routing Entry (ADDRTGE)

Type choices, press Enter.

Subsystem description  . . . . .   QCMN          Name
  Library  . . . . . . . . . . .     *LIBL       Name, *LIBL, *CURLIB
Routing entry sequence number  .   1             1-9999
Comparison data:
  Compare value  . . . . . . . .   CCIN

  Starting position  . . . . . .   37              1-80
Program to call  . . . . . . . .   AEGISRTR      Name, *RTGDTA
  Library  . . . . . . . . . . .     *LIBL       Name, *LIBL, *CURLIB
Class  . . . . . . . . . . . . .   QINTER        Name, *SBSD
  Library  . . . . . . . . . . .                 Name, *LIBL, *CURLIB
Maximum active routing steps . .   *NOMAX        0-1000, *NOMAX
Storage pool identifier  . . . .   1             1-10



                                                                      Bottom
F3=Exit  F4=Prompt  F5=Refresh  F12=Cancel  F13=How to use this display
F24=More keys
```

Figure 13. The ADDRTGE command screen

Automatic configuration of dynamic devices

Automatic configuration enables OS/400 to accept an incoming call from a remote APPC system (including another iSeries system or a personal computer), even though no varied-on controller description matches the LAN address of the calling system. OS/400 automatically creates a controller description for an APPN-capable APPC controller. You can set up intercommunication for dynamic devices by providing a model controller definition for each line on which an incoming request can be received for a CICS/400 system.

Controlling automatic configuration

OS/400 controls automatic configuration on a line-by-line basis. The incoming call must be on a line defined with AUTOCRTCTL(*YES)[9]. The operator can change the AUTOCRTCTL parameter in a token-ring or Ethernet line description at any time, without varying off any controllers that are attached to the line.

Automatic-configuration parameters

For automatic configuration of controllers, OS/400 derives parameters from three sources as follows:

- The initial XID exchange between OS/400 and the calling system
- A model controller
- The system-supplied defaults as applied to manually-configured controllers

9. The system QAUTOCFG value does not affect automatic LAN configuration.

XID exchange

The parameters derived from the XID exchange are:

RMTNETID

RMTCPNAME

ADPTADR

SSAP

DSAP

NODETYPE

TMSGRPNBR
Set to *CALC

CPSSN

SWTLINLST
Set to the line on which the call was received

If the caller supplies a control-point name in its XID, NODETYPE is set to *CALC and CPSSN is set to *YES. If the caller does not supply a control-point name in its XID, NODETYPE is set to *LENNODE and CPSSN is set to *NO. In an automatically created controller description, SWTLINLST contains only one line.

Model controller

If a model controller[10] specifies the line on which the call was received in its SWTLINLST parameter, OS/400 uses the model for parameters not derived from the XID exchange.

System defaults

System defaults provide any parameters not supplied by the XID exchange or a model controller description.

TCP/IP Connectivity for Client

TCP/IP connectivity is supported in CICS/400 by providing a TCP/IP listener for client requests sent over TCP/IP protocols. This listener is automatically started by the CICS control region based on the setting for the TCPPORT parameter in CICS system initialization table (SIT). If a TCP/IP port number is specified in the SIT, CICS will start the listener which will direct CICS client communications into the CICS control region.

The TCP/IP support for CICS/400 uses APPC communications internally to communicate with user transactions. Therefore, it is required to configure APPC as previously described. Specifically, the following are required:

- Ensure that the Application (APPLID) parameter of the SIT table of the Control Region is the same as the name of the Control Region.
- Define the TCP/IP port number in range between 1 through 65535 in the TCPPORT parameter. The TCP/IP port number must be the same as the one used by all clients connecting to this control region. Ensure the port is not already in use with NETSTAT option 3. A number above 1024 will avoid conflicts with the well-known ports.

10. A model controller specifies MDLCTL (*YES).

```
CHGCICSSIT  LIB(control_region_library)
            GROUP(control_region_group)
            APPLID(control_region_name)
            TCPPORT(port_number)
```

- Add a communication entry for the prestarted jobs in the subsystem description.

  ```
  ADDCMNE  SBSD(subsystem_library/subsystem_description)
           DEV(device_name)  MODE(CICSTCP)
  ```

  CICS/400 will automatically create controller descriptions and device descriptions for its internal communication. This communication entry requires a device name in Device (DEV) parameter that is of the form of the Control Region name plus an asterisk character. For example if the Control Region name is ABC1, the value in Device parameter must be ABC1*.
- Ensure that a prestart job entry exists in the subsystem description for the Control Region subsystem. You should also specify the number of AEGISICC jobs that will be used for these transactions (initial, additional and maximum number of jobs to start).

  ```
  ADDPJE  SBSD(subsystem_library/subsystem_description)
          PGM(QCICS/AEGISICC)
          INLJOBS(initial_num_jobs)
          ADLJOBS(additio_num_jobs)
          MAXJOBS(maxium_num_jobs)
  ```
- Add a TCS entry in the Control Region (with ADDCICSTCS command). This ensures that the prestarted jobs are automatically prestarted when the CICS Control Region is started. If you do not do this, the prestarted jobs must be started manually. Note that as in APPC connectivity, prestarted jobs cannot be started when the subsystem is started, they must be started after the CICS control region is operational.

  ```
  ADDCICSTCS  LIB(control_region_library)
              GROUP(control_region_group)
              SYSID(XXXX)  SNDPFX(S1)  SNDLMT(10)
              RCVPFX(R1)  RCVLMT(10)
  ```
- Start the Control Region. CICS/400 will start the TCP/IP listener in the same subsystem as the other control region jobs and will ensure that APPC Controller Descriptions and APPC Device Descriptions are created as required.

The TCP/IP listener is implemented with a set of jobs to monitor for TCP/IP communications. If you observe problems with these communications you can check the following places for information related to the operations of the TCP/IP listener function:

- Subsystem joblog
- History lob
- Joblogs in the AEGWPWKR and AEGWPSSN jobs

Chapter 4. Distributed program link

Using the EXEC CICS LINK command, a CICS application program can pass control to a second program, which runs as a called subroutine, returning control on completion to the point of invocation in the first program. Distributed program link extends the use of the EXEC CICS LINK command so that the linking and linked-to programs can be in different CICS systems. From a CICS/400 application program, you can link to a program running in any CICS family product. In most CICS family products, an application program can link to a CICS/400 program (see Table 1 on page 6).

Two ways to use DPL

An application program can use DPL in two ways, either ignoring the location of the linked-to program or explicitly specifying a remote system name.

Ignoring the location of resources

An application that uses DPL need not know the location of the linked-to program; it can issue an EXEC CICS LINK command as if the program is owned by the local system. In a CICS processing program table (PPT) entry, the system programmer can specify that the named program is owned not by the local system but by a remote system.

Explicitly specifying the remote system

On an EXEC CICS LINK command, an application program can use the SYSID option to specify the name of the TCS table entry (SYSID parameter of the ADDCICSTCS command) that defines the connection to the remote system that owns the program. The request is routed directly to the system named in the SYSID option, and the local resource definition tables are not used. Using the SYSID option in this way destroys the local program's independence of the linked-to program's location. The advantage is that ***any*** system, including the local system, can be named in the SYSID parameter of the TCS. The decision whether to access a local or remote program can be taken at execution time, based on parameters passed to the application program.

Serial connections

A PPT definition of the linked-to program is required in the remote CICS system to which the DPL request is directed. This definition may itself be a remote definition, causing the request to be relayed to another CICS system. When this occurs, the linking system and the linked-to system are said to be serially connected.

Synchronization and data integrity

In general, the linking program initiates commitment in both systems at task end. In CICS/400, CICS for OS/2, and CICS for OS/390, this can be varied by coding the SYNCONRETURN option in the EXEC CICS LINK command.

If you code SYNCONRETURN in the EXEC CICS LINK command, the linked-to program commits its resource updates immediately before returning control to the linking program. There are separate logical units of work (LUWs) in the two

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

Determining how a program was started

A linked-to program can determine whether it was invoked with the SYNCONRETURN option. In the EXEC CICS ASSIGN command, use the STARTCODE option to specify a 2-byte field, in which one of the following values is returned:

D The program was started by a DPL request without a SYNCONRETURN option. The program cannot issue I/O requests against its principal facility, and cannot issue syncpoint requests.

DS The program was started by a DPL request with a SYNCONRETURN option. The program cannot issue I/O requests against its principal facility, but can issue syncpoint requests.

BDAM files, and IMS, DL/I, and SQL databases

DPL enables a CICS/400 application program to access BDAM files and IMS, DL/I, and SQL databases in a CICS/mainframe system. The CICS/400 program links to a CICS/mainframe application program that reads and updates the databases or files.

Restrictions on programs invoked by DPL

There are certain restrictions on programs that are invoked by DPL.

Because no terminal is involved, linked-to programs should not issue:

- Terminal control commands to the system in which the linking program is running
- Commands that inquire on terminal attributes
- BMS commands
- Commands that address the TCTUA

Because of the TCTUA restriction, a linked-to program should use the COMMAREA to pass data. For guidance in optimizing the use of the COMMAREA, see "Performance optimization for DPL" on page 63.

Note: When DB2® data (on CICS/ESA or CICS/MVS) or SQL data (on CICS/VSE) is accessed from a CICS/400 transaction, security access is based on the transaction identifier that CICS/400 passes to the mirror transaction in CICS/mainframe. The linked-to program's EIBTRANID is set to the transaction identifier passed from CICS/400 and this is used for the duration of the link. If no transaction identifier is passed, EIBTRANID specifies the default mirror for the CICS/mainframe product.

Restricting a program to the DPL subset

In the program definition (ADDCICSPPT) command (63), enter **APISET(*DPLSUBSET)** to specify that a program, when invoked by an EXEC CICS LINK command, is restricted to the DPL subset of the API. This enables the program to be tested by a local EXEC CICS LINK command before it is invoked remotely. The use of prohibited commands are then detected before the program is used in a production environment.

Abends when using DPL

If the linked-to CICS program terminates abnormally and does not handle the abend itself, the mirror program returns an abend code. The code returned is that which would have been returned by an EXEC CICS ASSIGN ABCODE command. Note that the abend code returned to the linking CICS system represents the last abend to occur in the mirror program, which may have handled other abends before terminating.

Performance optimization for DPL

The performance of DPL may be affected by the amount of data transmitted, which includes the optional COMMAREA specified in an EXEC CICS LINK command. The length of the COMMAREA is in the range 1-32 767 bytes.

CICS/400 and the other CICS products contain algorithms designed to reduce the number of bytes to be transmitted. The algorithms remove some trailing binary zeros from the COMMAREA before transmission and restore them after transmission. The operation of these algorithms is transparent to the application programs, which always see the full-size COMMAREA.

When transmission time accounts for a significant part of the response time at a user terminal or workstation, application programs may be able to improve performance by using the DATALENGTH option in the EXEC CICS LINK command. This option specifies a contiguous area of storage, at the start of the COMMAREA, to be passed to the invoked program. For example, if all the data to be transmitted is grouped in the first 100 bytes of a 30 000-byte COMMAREA, and DATALENGTH(100) is specified, only the first 100 bytes are transmitted, although the whole COMMAREA is returned by the invoked program.

Why use DPL?

The following are some reasons why you might use DPL:

- To separate the end-user interface from the application business logic
- To obtain performance benefits from running programs closer to the resources they access
- To provide a simpler solution than distributed transaction processing (DTP)

Resource definition

The *CICS for iSeries Administration and Operations Guide* gives a full description of all CICS/400 resource definition commands. This section contains guidance about parameters that are important in the definition of resources to support DPL.

A remote program definition entry is required for each remote program that is to be invoked by DPL. If a DPL command or the definition of the invoked program names a transaction in the TRANSID parameter, the named transaction must be defined in the remote system that executes the program and must invoke that system's mirror program.

Program definition, ADDCICSPPT

The Add CICS/400 Processing Program Table (ADDCICSPPT) command adds an entry to the processing program table (PPT). Each entry contains information about a CICS/400 program. The following is guidance information about parameters that

are relevant to the definition of a remote program linked-to by DPL. For full details, see the *CICS for iSeries Administration and Operations Guide*.

CICS Program (PGMID)
Specifies the local name of the remote program. This is the name used in a local EXEC CICS LINK command that invokes the program.

API commands (APISET)
Specifies whether the program should execute only commands that are in the DPL subset of the API (see "Restricting a program to the DPL subset" on page 62). If **APISET(*DPLSUBSET)** is specified, the execution of a restricted command[11] raises the INVREQ exception condition (RESP2=200) or causes program termination with abend code ADPL.

This parameter enables local testing of a program that is to be invoked by DPL. See "Restrictions on programs invoked by DPL" on page 62.

CICS system (SYSID)
Specifies the CICS system in which the program is located. This is the name of the TCS table entry (SYSID parameter of the ADDCICSTCS command) that defines the connection to the remote system.

Remote CICS program (RMTPGMID)
Specifies the name of the program in the remote CICS system in which it is located. **RMTPGMID(*PGMID)** means that the local and remote names are the same.

Transaction (TRANSID)
Specifies a remote transaction under which this program is executed when it is invoked by a CICS distributed program link request from the local CICS/400 system. The environment in which the program runs is determined by the attributes of this transaction, which must invoke the same mirror program as the remote mirror transaction.

TRANSID(*NONE) means that no overriding transaction is used, and that the program runs under the remote mirror transaction.

Example
This command adds a PPT entry, called LOCLPGM, to a CICS/400 group, called GROUP, in an OS/400 library called CICSWORK.

```
ADDCICSPPT LIB(CICSWORK) GROUP(GROUP) PGMID(LOCLPGM) SYSID(CONR)
           RMTPGMID(REMTPGM) TRANSID(RTRN)
```

When the EXEC CICS LINK PROGRAM(LOCLPGM) command is executed locally, a link request is shipped on connection CONR to a remote CICS system, in which program REMTPGM is executed using transaction RTRN. RTRN must invoke the same mirror program as the remote mirror transaction.

Transaction definition, ADDCICSPCT
If a DPL command or the definition of the invoked program names a transaction in the TRANSID parameter, the named transaction must be defined in the remote system that executes the program and must invoke that system's mirror program.

The Add CICS/400 Program Control Table (ADDCICSPCT) command adds an entry to the CICS/400 program control table. Each entry contains information about a CICS transaction. The following is guidance information for defining a

11. See "Restrictions on programs invoked by DPL" on page 62.

transaction that is named in the TRANSID parameter of an incoming DPL request. For full details, see the *CICS for iSeries Administration and Operations Guide*.

Transaction (TRANSID)
Specifies the name of a local transaction. If this name is specified in the TRANSID parameter of an incoming DPL command, the attributes of this transaction define the environment in which the linked program is executed.

Program (PGMID)
Specifies the program to be executed under control of the transaction being defined. If the transaction is to be invoked in the TRANSID parameter of an incoming DPL request, this parameter must specify the local mirror program. In a CICS/400 system, specify **PGMID(AEGFSMIR)**.

CICS system (SYSID)
Specifies the CICS system in which the transaction is executed, therefore code **SYSID(*NONE)**.

Example

This command adds a PCT entry, called RTRN, to a CICS/400 group, called GROUP, in an OS/400 library called CICSWORK.

```
ADDCICSPCT LIB(CICSWORK) GROUP(GROUP) TRANSID(RTRN) PGMID(AEGFSMIR)
           SYSID(*NONE)
```

In terms of the ADDCICSPPT example in "Program definition, ADDCICSPPT" on page 63, this PCT entry is in the remote system. Transaction RTRN is defined to support incoming DPL requests that require transaction attributes other than those of the local mirror transaction. This transaction ***must*** invoke the mirror program, AEGFSMIR. To use this transaction, an incoming DPL request specifies TRANSID(RTRN), either in the EXEC CICS LINK command, or in the PPT entry for the invoked program, as in the ADDCICSPPT example.

Chapter 5. Function shipping

CICS function shipping enables CICS application programs to:

- Access CICS files owned by other CICS systems.
- Transfer data to or from transient data and temporary storage queues in other CICS systems.
- Initiate transactions in other CICS systems. (This form of communication is described in Chapter 7, "Asynchronous processing" on page 79.)

Note: Function-shipped commands cannot access BDAM files, or IMS, DL/I, or DB2 databases in a CICS/mainframe system. To access this data, use DPL (see "BDAM files, and IMS, DL/I, and SQL databases" on page 62).

Two ways to use function shipping

An application program can use function shipping in two ways, either ignoring the location of resources or explicitly specifying a remote system name.

Ignoring the location of resources

An application that uses function shipping need not know the location of the requested resources; it can issue commands as if all resources are owned by the local system. Entries in the CICS resource definition tables (see 68) allow the system programmer to specify that the named resource is owned not by the local system but by a remote system.

Explicitly specifying the remote system

In a resource-accessing command, an application program can use the SYSID option of EXEC CICS commands to specify the name of the TCS table entry (SYSID parameter of the ADDCICSTCS command) that defines the connection to the remote system that owns the resource. The request is routed directly to the system named in the SYSID option, and the local resource definition tables 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 parameter of the TCS entry.

Serial connections

A definition of the resource being accessed is required in the remote CICS system to which the function-shipping request is directed. This definition may itself be a remote definition, causing the request to be relayed to another CICS system. When this occurs, the linking system and the linked-to system are said to be serially connected.

CICS file control data sets

Function shipping allows read and update access to files located on a remote CICS system. The EXEC CICS INQUIRE FILE, INQUIRE DSNAME, SET FILE, and SET DSNAME commands are not supported.

Note: Care should be taken when designing systems that use remote file requests containing physical record identifier values (for example, VSAM RBA files,

and files with keys not embedded in the record). Application programs in remote systems must have access to the correct values following the updating or reorganization of such files.

Transient data

When an application program accesses intrapartition or extrapartition transient data queues on a remote system, the queue definition in the remote system specifies whether the queue is protected, 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 be executed 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.

Local and remote names

Any type of remote resource can be defined with a local name that is different from its name in its owning system (see "Resource definition"). This is useful when resources in different systems have the same name. For example, a program can send data to the CICS service destinations, such as the CSMT message log, in both local and remote systems.

Synchronization

The OS/400 syncpoint services ensure that when the requesting transaction reaches a syncpoint, any mirror transactions that are updating protected resources also take a syncpoint, using sync level 2 protocols, so that changes to protected resources in remote and local systems are consistent.

Data security and integrity

Protection of data accessed by function shipping is the responsibility of the file-owning system.

A resource update caused by a function shipping request is committed when the request-issuing program issues a syncpoint request or terminates successfully.

Resource definition

The *CICS for iSeries Administration and Operations Guide* gives a full description of all CICS/400 resource definition commands. This section contains guidance about parameters that are important in the definition of remote data resources (files, transient data destinations, and temporary storage queues) that are to be accessed by function shipping.

File definition, ADDCICSFCT

The Add CICS/400 File Control Table (ADDCICSFCT) command adds an entry to the CICS/400 file control table. The following is guidance about parameters that are important when defining remote files to be accessed by function shipping. For full details, see the *CICS for iSeries Administration and Operations Guide*.

CICS file (FILEID)
Specifies the local name of the remote file. This is the name used in a local function shipping request.

CICS system (SYSID)
Specifies the local name of the remote CICS system. This is the name of a local TCS table entry (SYSID parameter of the ADDCICSTCS command) that defines the connection to the remote system.

Remote CICS file (RMTFILE)
Specifies the name of the file in the remote CICS system. If the remote system is a CICS/400 system, this is the name in the FILEID parameter of the ADDCICSFCT definition in the remote system. **RMTFILE(*FILEID)** means that the remote name is the same as the local name specified in the FILEID parameter of this command.

Remote maximum key length (RMTKEYLEN)
Specifies the length in bytes of the key field. This value must be the same as that specified in the description of the file in the remote system in which it is ***locally*** defined.

The definition of the file in the system named in the SYSID parameter may itself be a remote definition; there may be a chain of remote definitions. Find the system in which the file has a local definition. If this system is a CICS/400 system, examine the FILE and MBR parameters of the local ADDCICSFCT definition. These parameters contain the name and location of the OS/400 database file. The required key length is in the OS/400 description of the file or in the data description specification (DDS) from which it was created.

If the file records have no keys, specify **0**.

Remote maximum record length (RMTLENGTH)
Specifies the maximum record size in bytes. This value must be the same as that specified in the description of the file in the remote system in which it is ***locally*** defined.

As described above for RMTKEYLEN, find the system in which the file has a local definition. If that system is a CICS/400 system, the required record length is in the OS/400 description of the file or in the data description specification (DDS) from which it was created.

Example
This command adds an FCT entry, called LOCLFILE, to a CICS/400 group, called GROUP, in an OS/400 library called CICSWORK.

```
ADDCICSFCT LIB(CICSWORK) GROUP(GROUP) FILEID(LOCLFILE) SYSID(CONR)
           RMTFILE(REMTFILE) RMTKEYLEN(6) RMTLENGTH(86)
```

This entry enables the function shipping of an access request for file LOCLFILE over connection CONR to a remote CICS system, where it accesses a file REMTFILE, which has key length 6 and record length 86.

Transient data queue definition, ADDCICSDCT
The Add CICS/400 Destination Control Table (ADDCICSDCT) command adds an entry to the CICS/400 destination control table, which defines transient data (TD) queues. The following is guidance about parameters that are important when defining remote TD queues to be accessed by function shipping. For full details, see the *CICS for iSeries Administration and Operations Guide*.

Destination (DEST)
Specifies the local name of the remote transient data queue.

Type (TYPE)
Specifies the type of transient data queue. For a remote transient data queue, code **TYPE(*REMOTE)**.

Remote destination (RMTDEST)
Specifies the name of the transient data queue in the remote CICS system in which it is locally defined. If the remote system is a CICS/400 system, this is the name in the DEST parameter of the local ADDCICSDCT definition. **RMTDEST(*DEST)** means that the remote name is the same as the local name specified in the DEST parameter.

CICS system (SYSID)
Specifies the local name of the remote CICS system in which the transient data queue is located. This is the name of the TCS table entry (SYSID parameter of the ADDCICSTCS command) that defines the connection to the remote system.

Example
This command adds an DCT entry, called LTDQ, to a CICS/400 group, called GROUP, in an OS/400 library called CICSWORK.

```
ADDCICSDCT LIB(CICSWORK) GROUP(GROUP) DEST(LTDQ) SYSID(CONR)
           RMTDEST(RTDQ) TYPE(*REMOTE)
```

This entry enables the function shipping of an access request for transient data queue LTDQ over connection CONR to a remote CICS system, where it accesses TD queue RTDQ.

Temporary storage queue definition, ADDCICSTST
The Add CICS/400 Temporary Storage Table (ADDCICSTST) command adds an entry to the CICS/400 temporary storage table. Each entry contains information about temporary storage (TS) queues that need special processing. For example, a TST entry is required when the queue resides on another CICS system.

CICS/400 application programs use CICS/400 temporary storage queues to store data for later retrieval. The entry can be defined using the entire queue name or the prefix of a generic queue name. When processing a remote queue with a name that begins with the same characters (generic or entire) as the name of a local TST definition, CICS/400 uses the attributes in that definition. The *CICS for iSeries Administration and Operations Guide* contains a full description of all the parameters of the ADDCICSTST command.

The following is guidance about parameters that are important when defining remote TS queues to be accessed by function shipping.

Queue (TSQUEUE)
Specifies the local name (generic or entire) of the TS queue.

Type (TYPE)
Specifies the type of the TS queue. For a remote temporary storage queue, code **TYPE(*REMOTE)**.

CICS system (SYSID)
Specifies the local name of the remote system in which the temporary storage queues associated with this definition are located. This is the name of the local TCS table entry (SYSID parameter of the ADDCICSTCS command) that defines the connection to the remote system.

Remote queue name (RMTQUEUE)
The temporary storage prefix that is used by the remote system in which the queues are located.

RMTQUEUE(*TSQUEUE) indicates that the remote prefix is the same as the local prefix specified in the TSQUEUE parameter of this command.

Example

This command adds a TST entry, called LOCLTSQ, to a CICS/400 group, called GROUP, in an OS/400 library called CICSWORK.

```
ADDCICSTST LIB(CICSWORK) GROUP(GROUP)
  TSQUEUE(LOCLTSQ) SYSID(CONR) RMTQUEUE(REMTTSQ)
```

This entry enables the function shipping of an access request for temporary storage queue LOCLTSQ over connection CONR to a remote CICS system, where it accesses TS queue REMTTSQ.

Chapter 6. Transaction routing

In transaction routing, an end user enters a remote transaction identifier at a terminal and the transaction runs as though it were owned by the local system. Transaction routing depends entirely on resource definition; no application programming is required.

Two resources are involved in transaction routing:

- The initiating terminal
- The initiated transaction

Terminal

The application-owning region (AOR) requires a definition of the remote terminal. This definition must contain the essential information needed to implement transaction routing, as follows:

- The local name of the terminal
- The name of the connection to the terminal-owning region (TOR)
- The name of the terminal in the TOR

Additionally, the remote definition of a terminal must have the following characteristics:

- A subset of the 3270 extended data stream is supported.
- The color, highlight, PS, and outline extended attributes are supported but only to the extent that they are generated by BMS. (A field may have these attributes but an individual character within a field cannot have separate attributes.)
- The USRARASIZE parameter must be the same as the corresponding parameter in the remote system's TCT definition for the terminal.

There is an alternative to the creation of a remote definition of a terminal in any remote system to which it wants to direct a transaction routing request. This is to make the terminal's local definition shippable by coding **SHIP(*YES)**.

If a terminal definition is shippable, sufficient data is passed with a transaction routing request to enable the remote system to dynamically install the necessary remote terminal definition. For further information about shippable terminals, refer to the *CICS for iSeries Administration and Operations Guide*.

Transaction

The terminal-owning region requires a definition of the remote transaction. This definition specifies only information that is needed to implement transaction routing, as follows:

- The local name of the transaction
- The name of the connection to the application-owning region (AOR)
- The name of the transaction in the AOR

All other transaction characteristics are defined in its local definition in the application-owning region (AOR).

Serial connections

A definition of the transaction being accessed is required in the remote CICS system to which the transaction routing request is directed. This definition may itself be a remote definition, causing the request to be relayed to another CICS system. When this occurs, the linking system and the linked-to system are said to be serially connected.

If terminal definitions are shipped across a serial connection, the application-owning region (AOR) requires an indirect connection to the terminal-owning region (TOR). See the INDSYS parameter of the ADDCICSTCS command 40.

Resource definition

The *CICS for iSeries Administration and Operations Guide* gives a full description of all CICS/400 resource definition commands. This section contains guidance about parameters that are important in the definition of transactions and terminals to support transaction routing.

Transaction definitions, ADDCICSPCT

The Add CICS/400 Program Control Table (ADDCICSPCT) command adds an entry to the CICS/400 Program Control Table. Each entry contains information about a CICS/400 transaction.

The following is guidance about parameters that are important when defining a remote transaction that is to be invoked by transaction routing. For full details, see the *CICS for iSeries Administration and Operations Guide*.

Transaction (TRANSID)
Specifies the local name of the transaction.

CICS system (SYSID)
Specifies the local name of the application-owning region (AOR). This is the name of the TCS table entry (SYSID parameter of the ADDCICSTCS command) that defines the connection to the AOR.

Remote transaction (RMTTRANSID)
Specifies the name of the transaction in the remote application-owning region (AOR). **RMTTRANSID(*TRANSID)** means that this name is the same as the local name specified in the TRANSID parameter of this command.

Example

This command adds a PCT entry, called LTRN, to a CICS/400 group, called GROUP, in an OS/400 library called CICSWORK.

```
ADDCICSPCT LIB(CICSWORK) GROUP(GROUP) TRANSID(LTRN) SYSID(CONR)
           RMTTRANSID(RTRN)
```

This entry causes a request for transaction LTRN to be routed over connection CONR to a remote CICS system, in which transaction RTRN is executed.

Terminal definitions, ADDCICSTCT

The Add CICS/400 Terminal Control Table (ADDCICSTCT) command adds an entry to the CICS/400 Terminal Control Table (TCT). Each entry contains information about a CICS/400 terminal (display device, model, or printer).

To support transaction routing, this command is used in the application-owning region (AOR) to create a remote definition of the terminal from which the transaction is invoked. The following is guidance about parameters that are important when defining a remote terminal. For full details, see the *CICS for iSeries Administration and Operations Guide*.

CICS device (CICSDEV)
Specifies the local name of the terminal.

CICS system (SYSID)
Specifies the local name of the terminal-owning region (TOR). This is the name of the TCS table entry (SYSID parameter of the ADDCICSTCS command) that defines the connection to the remote system, and can be an indirect connection[12] . This connection is used to route transaction output from this system (the AOR) back to the terminal in the remote system (the TOR).

Remote CICS device (RMTDEV)
Specifies the name of the terminal in the terminal-owning region (TOR). **RMTDEV(*CICSDEV)** means that the remote name is the same as the local name specified in the CICSDEV parameter of this command.

User area size (USRARASIZE)
Specifies the length of the terminal user area. This value should be the same as that specified in the USRARASIZE parameter of the terminal's definition in its owning region (the TOR).

Local definition of shippable terminal

To avoid the need for a remote definition in the application-owning region, a terminal can be defined as shippable in its local definition in the terminal-owning region.

Ship to another CICS system (SHIP)
Specifies whether the terminal definition can be shipped to another CICS system. If the terminal is to be shipped, code **SHIP(*YES)**.

Example

This command adds a TCT entry, called REMTTERM, to a CICS/400 group, called GROUP, in an OS/400 library called CICSWORK.

```
ADDCICSTCT LIB(CICSWORK) GROUP(GROUP) CICSDEV(REMTTERM) SYSID(CONL)
           RMTDEV(LOCLTERM)
```

This entry is installed in an application-owning system (AOR) that receives transaction routing requests from a remote terminal LOCLTERM. The entry enables routing of data from the AOR across connection CONL back to the terminal-owning region (TOR).

Inbound transaction routing to the CEMT transaction

In CICS/400, invocation of the CEMT transaction leads to a pseudoconversation with repeated executions of CEMT. This complicates transaction routing to CEMT in CICS/400. The problem can occur in transaction routing to a pseudoconversation in any CICS product. Figures 14 and 15 illustrate the problem and the suggested solution.

In Figure 14, AEMT is entered at a terminal in System A to invoke the CEMT transaction in System B. CICS/400 in System B runs the first execution of CEMT

12. See INDSYS parameter of ADDCICSTCS command 40.

and returns the value “CEMT” in TRANSID. The next input at the terminal causes the invocation of CEMT in System A, rather than in System B as intended.

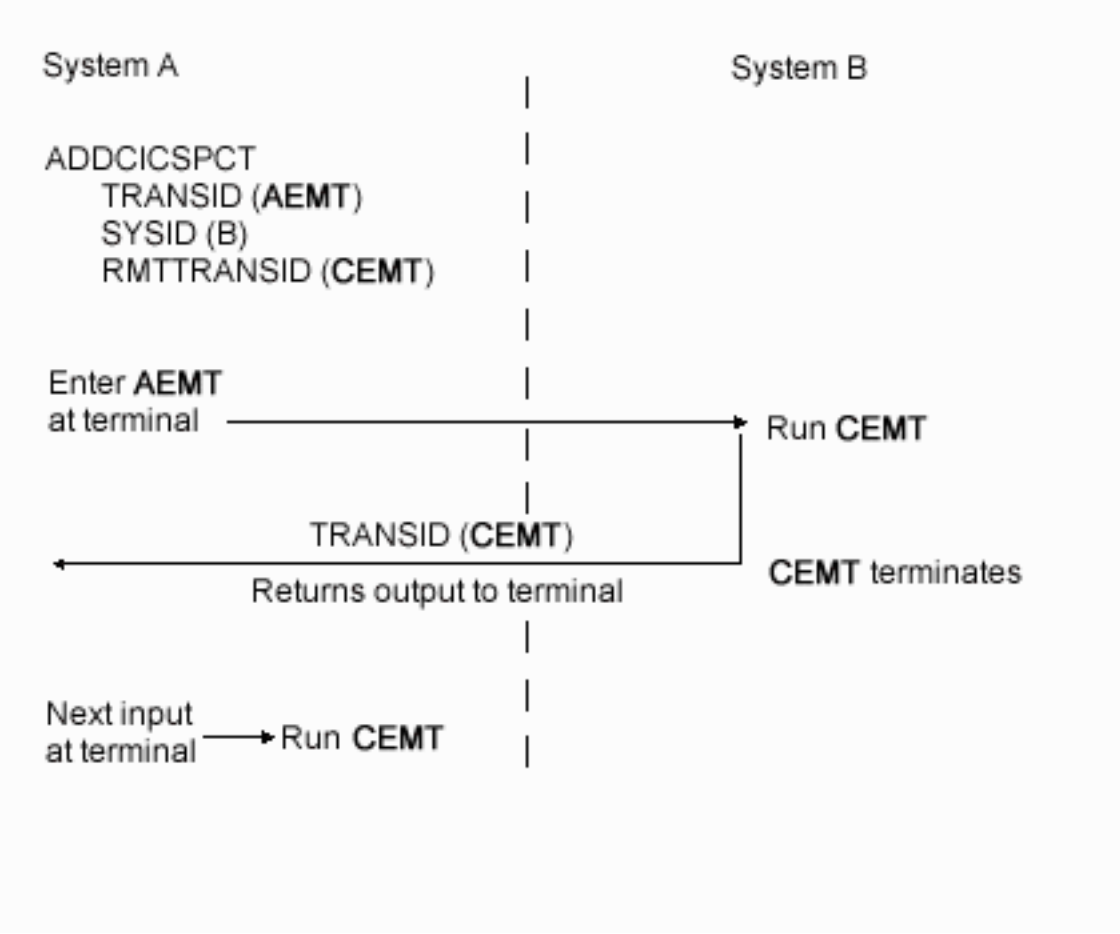


Figure 14. Transaction routing to the CEMT transaction–the problem

In Figure 15 on page 77, the remote transaction definition in System A specifies the same local and remote name AEMT. In System B, transaction AEMT is defined to invoke the master terminal program, AEGCMDRV, so that transaction AEMT is equivalent to CEMT. Now, AEMT is entered at a terminal in System A to invoke the AEMT transaction in System B. CICS/400 in System B runs the first execution of AEMT and returns the value “AEMT” in TRANSID. The next input at the terminal causes the invocation of AEMT in System B, as intended.

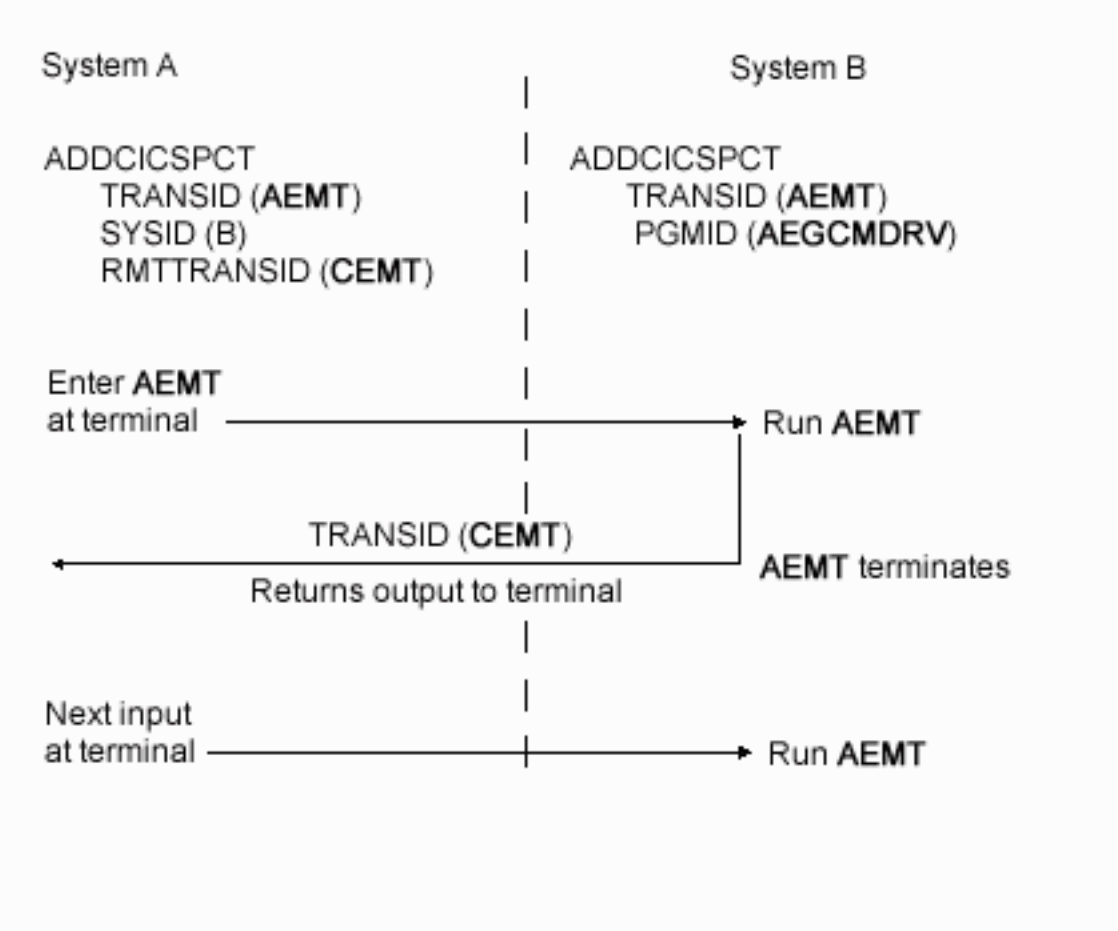


Figure 15. Transaction routing to the CEMT transaction–the solution

Transaction routing to a pseudoconversation

The CEMT transaction in CICS/400, just discussed, is one example of a pseudoconversation–any pseudoconversation can contain a mixture of local and routed transactions. A problem arises if identical transaction names exist in different systems. The following procedure applies to all cases, and, where necessary, forces redefinition of transactions in the application-owning system as in the CEMT example.

Procedure for routed transactions in a pseudoconversation

In the terminal-owning system, define each routed transaction in a pseudoconversation with identical local and remote names, and as residing in the application-owning system in which it is locally defined.

Chapter 7. Asynchronous processing

Asynchronous processing is a special case of function shipping in which the shipped command starts a remote transaction. Unlike distributed transaction processing (DTP), the initiating and initiated transactions do not engage in synchronous communication. Instead, they are executed and terminated independently.

The interval control commands that can be used for asynchronous processing are:

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

Two ways to initiate asynchronous processing

Asynchronous processing is initiated by the issuing of an EXEC CICS START command. Like DPL, the application program can ignore the location of the started transaction or can explicitly specify the system name.

Ignoring the location of the transaction

A program can issue an EXEC CICS START command for a remote transaction as if the transaction is local. An entry in the program control table (PCT) defines the transaction to CICS and specifies the owning system, local or remote.

Explicitly specifying a remote system

On an EXEC CICS START command, an application program can use the SYSID option to specify the name of the TCS table entry (SYSID parameter of the ADDCICSTCS command) that defines the connection to the remote system that owns the transaction. The request is routed directly to the system named in the SYSID option, and the local resource definition tables are not used. Using the SYSID option in this way destroys the local program's independence of the started transaction's location. The advantage is that ***any*** system, including the local system, can be named in the SYSID parameter of the TCS. The decision whether to access a local or remote transaction can be taken at execution time, based on initialization options passed to the application program.

Note: Asynchronous processing can also be initiated by using distributed transaction processing (DTP), a cross-system method with no single-system equivalent. This is much more complicated than the use of the EXEC CICS START command, and it would be wasteful to use DTP simply to start a transaction. Asynchronous processing started by DTP is likely to be a by-product of a conversational application.

Starting and canceling remote transactions

The interval control EXEC CICS START command is used to queue a transaction-initiation request in a remote CICS system, to which the command is function-shipped. In the remote system, the mirror transaction (CVMI) is invoked to issue the EXEC CICS START command. You can include time-control information on the shipped EXEC CICS START command, using either the INTERVAL or the 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 can be in different time zones, use the INTERVAL option to specify an absolute time interval that is independent of time zones. The time interval, specified in the INTERVAL or TIME option of an EXEC CICS START command, determines ***the time at which the remote transaction is to be initiated***, not the time at which the request is to be shipped to the remote system.

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

Passing information with the EXEC CICS START command

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

- User data specified in the FROM option. This is the principal way in which data can be passed to the remote transaction.
- Temporary storage queue—named in the QUEUE option. This is an additional way of passing data. The queue can be in any CICS system that is accessible to the system on which the remote transaction is executed.
- 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, an automatic transaction initiation (ATI) request is sent to the terminal-owning region (TOR).
- A transaction name and an associated terminal name—specified in the RTRANSID and RTERMID options. These options enable the local transaction to specify transaction and terminal names for the remote transaction to use in an EXEC CICS START command to initiate a transaction in the local system.

Passing an applid with the EXEC CICS START command

If you have a transaction that can be started from several different systems, and that is required to issue an EXEC CICS 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 EXEC CICS START command. A transaction can obtain its own local applid by using an EXEC CICS 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 [13] option of the START command. When the connection between the two systems is through VTAM, this

13. CICS OS/2 does not support the NOCHECK option.

can result in considerably improved performance. The trade-off is between performance and sophisticated recovery procedures.

A typical use for the EXEC CICS START NOCHECK command is in the example 81.

Deferred sending of start requests with the NOCHECK option

For EXEC CICS START commands with the NOCHECK option, CICS defers transmission of the request until one of the following events occurs:

- The transaction issues a function shipping request for the same system.
- The transaction terminates (implicit syncpoint).

The first, or only, start request transmitted from a transaction to a remote system carries the begin-bracket indicator; the last, or only, request carries the end-bracket indicator. Also, if any of the start requests issued by the transaction specifies PROTECT, syncpoint coordination occurs after the last request. The sequence of requests is transmitted within a single SNA bracket and all the requests are handled by the same mirror task.

The NOCHECK option is always required when shipping of the EXEC CICS START command is queued pending the establishment of links with the remote system (see "Local queuing of start requests for remote transactions").

Local queuing of start requests for remote transactions

When a local transaction is ready to ship an EXEC CICS 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. Local queuing can be attempted for an EXEC CICS START NOCHECK command if the system name is valid but the system is not available. A system is defined as not available if the system is *out of service* when the request is initiated, or if an attempt to initiate a session to the remote system fails.

In either of the above situations, CICS queues an EXEC CICS START NOCHECK command for a remote transaction if two conditions are satisfied:

- The SYSID option is not coded in the EXEC CICS START command, and
- The local definition of the transaction specifies LCLQUEUE(*YES).

Local queuing should be used only for EXEC CICS START NOCHECK commands that represent time-independent requests. The delay implied by local queuing affects the time at which the request is actually started.

Example: online inquiry on a remote database

To check credit ratings, 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 tasks can be executing the remote transaction concurrently. The remote tasks send their replies by initiating a local transaction (possibly the initiating transaction) to deliver the output to the operator terminal. The replies may not arrive in the order in which the inquiries were issued; correlation between the inquiries and the replies must be made by means of fields in the user data.

The operator's request causes the attachment of a transaction that issues an EXEC CICS START NOCHECK command to start the inquiry transaction in a remote system. The EXEC CICS START command specifies the operator's terminal identifier. The transaction that issued the EXEC CICS START command now terminates, leaving the terminal available to receive the answer or initiate another request.

The remote transaction makes the requested inquiry on its local database, then issues an EXEC CICS START NOCHECK command to start a transaction on the originating system. This command passes back the requested data, together with the operator's terminal identifier. Again, only one message passes between the systems. The transaction that is now started in the originating system formats the data and displays 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 and returned with the corresponding reply, or all replies must be self-describing.

"Started transaction satisfying multiple start requests" on page 83 describes a technique that could be used in the remote transaction and in the local transaction that receives the replies.

Including start request delivery in a logical unit of work

The delivery of a start request to a remote system can be made part of a logical unit of work by specifying the PROTECT option on the EXEC CICS START command. The PROTECT option indicates that the remote transaction must not be scheduled until the initiating transaction has terminated successfully.

Successful completion of the transaction 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 has been or will be initiated.

The started transaction

A CICS transaction that is initiated by an EXEC CICS START command can get the user data and other information associated with the request by using the EXEC CICS RETRIEVE command.

Started transaction satisfying multiple start requests

In accordance with the normal rules for interval control, CICS queues a start request 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 active transaction can issue a further EXEC CICS RETRIEVE command to access the data associated with the queued request. 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. A long-running transaction can accept multiple inquiries from a terminal and ship start requests to a remote system. In the remote system, the first request causes a transaction to start. From time to time, the started transaction can issue EXEC CICS RETRIEVE commands to receive the data associated with further requests, the absence of further requests being indicated by the ENDDATA condition.

The WAIT option of the EXEC CICS RETRIEVE command can be used to put the transaction into a WAIT state pending the arrival of the next start request from the remote system. In the example on 81, EXEC CICS RETRIEVE WAIT could be used in both the remote transaction and the local transaction that receives the replies.

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 time-out interval.

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.

Resource definition

The *CICS for iSeries Administration and Operations Guide* gives a full description of all CICS/400 resource definition commands. This section contains guidance about parameters that are important in the definition of a remote transaction to support asynchronous processing.

Transaction definition, ADDCICSPCT

The Add CICS/400 Program Control Table (ADDCICSPCT) command adds an entry to the CICS/400 Program Control Table. Each entry contains information about CICS/400 transactions. For full details, see the *CICS for iSeries Administration and Operations Guide*.

Transaction (TRANSID)
Specifies the local name of the remote transaction.

CICS system (SYSID)
Specifies the local name of the remote CICS system in which the transaction is located. This is the name of the TCS table entry (created by an ADDCICSTCS command) that defines the connection to the remote system.

Remote transaction (RMTTRANSID)
Specifies the name of the transaction in the remote CICS system in which it is

located. **RMTTRANSID(*TRANSID)** means that the remote name of the transaction is the same as its local name, which is specified in the TRANSID parameter of this command.

Local system queuing (LCLQUEUE)
Specifies whether, when a session to the remote system is unavailable, queuing on the local system is required.

Example

This command adds a PCT entry, called LTRN, to a CICS/400 group, called GROUP, in an OS/400 library called CICSWORK.

```
ADDCICSPCT LIB(CICSWORK) GROUP(GROUP) TRANSID(LTRN)
           RMTTRANSID(RTRN) SYSID(CONR) LCLQUEUE(*YES)
```

This entry enables the function shipping of a start request for transaction LTRN across connection CONR to a remote CICS system in which transaction RTRN is initiated. Local queuing is supported for requests for transaction LTRN.

Chapter 8. Security

This chapter gives an overview of security considerations when CICS/400 systems are interconnected, connected to other CICS products, or connected to other compatible systems. It is assumed that the reader is familiar with security in a single CICS/400 system, as described in the *CICS for iSeries Administration and Operations Guide*.

In a single CICS system, security restricts user access to resources required by the user's job responsibilities. For interconnected systems, the same principles apply, but additional definitions are needed for local and remote connections.

Planning for intercommunication security

In a CICS/400 system, intercommunication security relates to incoming requests for access to CICS resources. The security problem with incoming requests can be framed as a question—is this resource access authorized, or should it be rejected?

There are four levels of CICS security, all of which are optional:

- Bind-time security
- Link security (not supported by CICS/400)
- User security
- Resource security

Bind-time security

Intercommunication begins with the establishment of a session between two systems. The connection request can be made by either system. The establishment of a session gives remote users potential access to a system's resources and is the first intercommunication security exposure. The level of security applied at this stage is called bind-time security, which is also known as SNA session security or location security.

Link security

Link security applies a single user profile to a remote system as a whole, and is supported by most other CICS products, but not by CICS/400. All security checks in CICS/400 are at the user and resource levels.

User security

User security (also known as conversation or attach security) is used to restrict remote-user access to the local system, and is applied when CICS/400 starts a transaction initiated from a remote system. User security is based on the user ID (and, optionally, password) transmitted by the remote system. For an incoming request to satisfy this level of security, the user ID and password (if present) must match an OS/400 user profile. User profiles are created by a CRTUSRPRF command.

Resource security

Resource security protects local resources such as files, databases, programs, and transactions, from unauthorized access by users, local or remote.

Implementing intercommunication security

This section describes implementation of the three levels of intercommunication security supported by CICS/400.

Bind-time security

An intercommunication session is bound on acceptance of a request to establish an APPC session with a remote system[14] . Binding a session involves negotiating session attributes that are compatible with the attributes of both binding partners. During the bind process, bind-time security is applied to ensure that an unauthorized system cannot bind a session to a CICS/400 system.

Bind-time security is optional, and can be specified only if it is supported by the remote system. CICS/400 and other members of the CICS family (except CICS OS/2) conform to the SNA session security architecture. To be eligible for bind-time security, a remote non-CICS system must conform to the same architecture.

The definition of a connection to a remote system assumes that all inbound requests on the connection are routed from that system and that all outbound requests on the connection are routed to that system. If a bind request is to succeed, both ends must hold the same bind password (or location password).

Bind password for intrasystem communication

For intrasystem communication, which is possible only between two CICS/400 systems, each system requires a bind password to be specified in any APPC device description (CRTDEVAPPC command) that defines a logical device that can be the source or recipient of a message. For a particular connection, the device descriptions at both ends must specify the same bind password, else the bind fails.

Bind password for CICS/400 intersystem communication

For intersystem communication, which is possible between CICS/400 and any CICS system (including another CICS/400 system), the bind password is specified in the APPN Remote Location Configuration List Entry that lists the APPN remote locations with which OS/400 can communicate. To set up a configuration list, use the CRTCFGL command for list type *APPNRMT.

For other CICS products, refer to the intercommunication documentation for information on where the bind password is defined.

The bind password is a 1-8 byte string (an even number of hexadecimal digits in the range 2-16).

User security

User security is applied when CICS/400 attaches a transaction that is initiated from a remote system. User security, subject to a check against the local OS/400 user profile for the remote user, grants the user access to the CICS/400 system. Access to individual resources is subject to a further check by resource (object) security.

Levels of user security

One of three levels of user security may apply:

14. The request can originate from either system.

1. No check is made and each user has the security profile of the default user specified for the remote location
2. A check is made on the user ID
3. A check is made on the user ID and password

Default user: The default user for a remote location is defined in the communications entry (ADDCMNE command) for the remote location in the communications subsystem. For information about the ADDCMNE command, refer to *Communications Management*.

Secure locations: A location can specify the amount of security information it requires from the other location in a remotely initiated CICS conversation. This is done to remove the need for applications to send passwords with user IDs, and to simplify network administration because passwords may not be the same on all systems. A location may be defined as a "secure location" or may default to a "non-secure location".

A secure location represents an acknowledgement by the target system that the security facilities of the source system are acceptable. The source location can then send an Already Verified Indicator (AVI) with the user ID because the target system will trust the security arrangements of the source system.

A non-secure location definition indicates that the target system wants the source system either to send both a user ID and a password or no security information at all, in which case the target system will use a default user profile for remotely initiated jobs.

Specifying the level of security: The level of security is defined by different commands depending on whether intrasystem or intersystem communication is being used:

- For intrasystem communication, the **Secure location** parameter in the APPC device description (CRTDEVAPPC) command.
- For intersystem communication, the **Secure location** parameter in the create configuration list (CRTCFGL) command that specifies the remote configuration list entry for the remote location.

In either case, **Secure location** can be specified as *YES or *NO.

Secure Location *YES
The user ID is checked against the OS/400 user profile for that user ID. An Already Verified Indicator is sent to show that the password for that user ID has been checked at the remote location.

A user profile must exist for the user ID.

Secure Location *NO
The OS/400 does not accept verification by the remote system. The remote (source) system may send both user ID and password, or neither of these. If neither is sent, the incoming request adopts the access authority of the default user (see "Default user"). If a user ID and password are sent, a user profile must exist for the user ID, and the password must match that of the user profile.

Resource security

When a transaction has been successfully attached, resource (or object) security determines which resources it can access. The transaction uses the authority of

either the remote user ID (**Secure Location *YES**) or the default user (**Secure Location *NO**). The OS/400 security officer grants object authority to each remote user ID in the same way as for a local user ID. For information on specifying object authority, see the *CICS for iSeries Administration and Operations Guide*.

Chapter 9. Data conversion

When communication takes place between a CICS/400 system and a remote CICS/6000 or CICS OS/2 system, conversion is necessary between the EBCDIC codes in CICS/400 and the ASCII codes in the remote CICS/6000 or CICS OS/2 system. Specifically, application-data conversion is necessary for function shipping, DPL, and distributed transaction processing. When transaction routing occurs in either direction between CICS/400 and CICS OS/2 or CICS/6000, data conversion is always done by CICS OS/2 or CICS/6000 (except for requests from CICS common client, see page 55). For further information, see *CICS Family: Interproduct Communication* and the Globalization topic in the iSeries Information Center.

Which system does the conversion?

In general, the ASCII system converts resource names and the receiving system converts application data. Table 5 shows where data conversion is done for function shipping and DPL. In the table, Receiver is the system that receives the request, and CVT is the CICS/400 conversion vector table, which contains an entry for each resource for which conversion is required. The table shows *all* the conversion automatically done in the CICS systems. In distributed transaction processing, the applications must initiate data conversion but can call the OS/400 conversion routine, QDCXLATE.

Table 5. Data conversion for function shipping and DPL

Request type	Data	Conversion type	Where converted
Temporary storage (*TSQUEUE in CVT, see "Distributed program link (DPL)" on page 3)	Queue name	Character	ASCII system
	IOAREA	As specified in CVT	Receiver
Transient data (*TDQUEUE in CVT, see "Distributed program link (DPL)" on page 3)	Queue name	Character	ASCII system
	IOAREA	As specified in CVT	Receiver
File control (*FILE in CVT, see "Distributed program link (DPL)" on page 3)	File name	Character	ASCII system
	IOAREA	As specified in CVT	Receiver
	Key	As specified in CVT	Receiver
Interval control (*START in CVT, see "Distributed program link (DPL)" on page 3)	Transid	Character	ASCII system
	FROM area	As specified in CVT	Receiver
	RTERMID, RTRANSID, REQID	Character	ASCII system
Program control (*LINK in CVT, see "Distributed program link (DPL)" on page 3)	Program name	Character	ASCII system
	COMMAREA	As specified in CVT	Receiver

Function shipping and DPL

For function shipping and DPL *from* CICS/400 *to* CICS for Open Systems or CICS for OS/2, the remote CICS system does all the required conversion.

For function shipping and DPL *to* CICS/400 *from* CICS for Open Systems or CICS for OS/2, the remote system converts resource names and CICS/400 converts user data areas.

The same rules apply to asynchronous processing initiated by an EXEC CICS START command, which is a special case of function shipping.

Serial connection

A receiving system recognizes the sending system's interchange code by the transmitted mirror-transaction name (CPMI for ASCII and CVMI for EBCDIC). When a serial connection[15] is used, user data is converted when the interchange code (ASCII or EBCDIC) changes between two systems in the serial path. Because of this, data conversion definitions may be required for a resource in a system other than its owning system.

For example, in Figure 16, a CICS OS/2 system **A** function ships a write request for a file **FILEA** to a CICS/400 system **C**. The request is sent across a serial connection through a second CICS/400 system **B**. "Function shipping" on page 4 shows that the data conversion requirements for **FILEA** are defined to system **B**, because that is where conversion takes place.

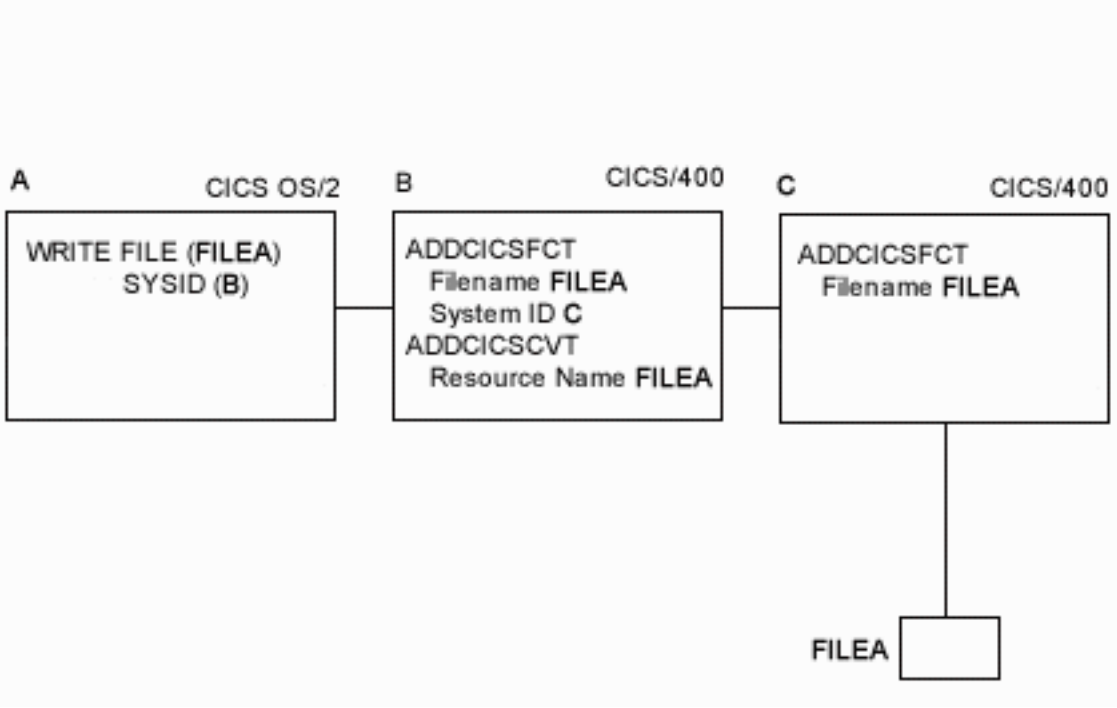


Figure 16. Function shipping. CICS OS/2 to CICS/400 with serial connection through CICS/400.

15. A serial connection was formerly called daisy chaining.

Distributed transaction processing

DTP uses application-specific data areas and cannot have a general procedure for data conversion. It is the application's responsibility to perform data conversion. Application design determines whether conversion is in CICS/400 or the remote CICS system.

Using the QDCXLATE function, a CICS/400 application program can link directly to the OS/400 conversion program. For details, refer to the *CL Programming* book.

Avoiding data conversion

Application design can reduce the amount of data conversion. For example, if a CICS/400 systems acts as a file manager for CICS OS/2 systems, the data in the file can be coded in ASCII, eliminating the need for data conversion.

If data is held at a CICS OS/2 workstation purely for the purpose of communicating with a CICS/400 system, the data can be coded in EBCDIC.

Types of Conversion

The possible types of conversion are:

Standard conversion
This applies to normal or single-byte character sets (SBCS), graphic or double-byte character sets (DBCS), mixed character sets (containing SBCS and DBCS data), and numeric data in Intell format.

No conversion
This applies to binary and packed decimal data. Treat other data as though it were binary if you do not want it converted.

Resource definition

The *CICS for iSeries Administration and Operations Guide* gives a full description of the CICS/400 resource definition commands for data conversion. This section contains guidance information for the important parameters of the ADCICSCVT command, which adds an entry to the CICS/400 conversion vector table (CVT). This guidance also applies to the same parameters when used to change or work with an entry (CHGCICSCVT and WRKCICSCVT commands).

Conversion Vector Table definition, ADDCICSCVT

The Add CICS/400 Conversion Vector Table (ADDCICSCVT) command adds an entry to the CVT. Each entry is associated with a specific resource and defines the conversion of user data transmitted to and from that resource during intercommunication between CICS/400 and either CICS OS/2 or CICS/6000. For full details, see the *CICS for iSeries Administration and Operations Guide*.

Required parameters

API command type (CMDTYPE)
Specifies the type of command for which this entry is used. Code **CMDTYPE(*FILE)**, **CMDTYPE(*TDQUEUE)**, **CMDTYPE(*TSQUEUE)**, **CMDTYPE(*START)**, or **CMDTYPE(*LINK)** for a file, transient data queue, temporary storage queue, transaction initiated by a START command, or program initiated by DPL, respectively.

Resource identifier (RSRCID)
Specifies the name of the resource, file (**CMDTYPE(*FILE)**), transient data

destination (**CMDTYPE(*TDQUEUE)**), temporary storage queue (**CMDTYPE(*TSQUEUE)**), transaction (**CMDTYPE(*START)**), or program (**CMDTYPE(*LINK)**).

Character identifier (CNVCHRID)
Specifies the code page and graphic character set to which or from which data is to be converted in transmissions between CICS/400 and a remote CICS system. This parameter is used in conjunction with the OS/400 system value QCHRID, which defines the local system code page and character set.

In the OS/400 library QUSRSYS, there are several conversion tables (objects of type *TBL). Data conversion requires two tables, one to convert from the remote code page and character set to the local code page and character set, and one to reverse the procedure.

The names of the tables to be used are derived from the CNVCHRID parameter and the system QCHRID value, as follows:

- To convert from remote to local code page and character set **Qaaadddccc**
- To converyt from local to remote code page and character set **Qcccbbbaaa**

where

aaa is the code page specified in CNVCHRID

bbb is the graphic character set specified in CNVCHRID

ccc is the code page specified in QCHRID

ddd is the graphic character set specified in QCHRID

If either table is not found in library QUSRSYS, message AEG7616 or AEG7617 is issued at startup or during dynamic installation.

If you cannot find tables in QUSRSYS that match your code page and graphic character set, you may find it useful to use a CNVCHRID of 999 999 and copy the standard QSYS/QASCII *TBL and QSYS/QEBCDIC *TBL into QUSRSYS and call them Qccc999999 and Q999dddccc respectively.

Optional parameters

Conversion information (CNVINF)
Specifies default conversion specifications for each user-data field. These specifications are used to convert data that does not satisfy any of the criteria specified in the SLTCTL parameter. For each field, this parameter gives:

- Offset from the beginning of the record
- Length
- Type of conversion
- SO/SI indicator

Note: When user data contains multiple record formats, use the SLTCTL parameter to select each type of record and specify its conversion specifications. If the SLTCTL parameter is coded, CICS/400 applies CNVINF specifications only to record types not selected by the SLTCTL criteria. The CNVINF parameter has 4 elements, which define a field and its conversion requirements. You can repeat the 4-element sequence up to 30 times to define up to 30 fields.

Starting location
Specifies the offset from the start of the record of the field to be converted. The default is 0 (field begins at the start of the record).

Type of conversion
Specifies the type of conversion. The possible values are:

***CHAR**
Character data. CICS/400 converts incoming data form the CNVCHRID code page to the OS/400 code page specified in the QCHRID system value.

***PACKED**
Packed decimal data. CICS/400 does not convert the data.

***BINARY**
Binary data. CICS/400 does not convert the data

***IDEOGRAPHIC**
Ideographic-based language. CICS/400 converts incoming data from the CNVCHRID-specified ideographic code page to the OS/400 code page specified in the OS/400 QCHRID system value.

***INTEL**
INTEL integer data. CICS/400 converts incoming INTEL data by reversing the bytes. The length of the field must be a 2 or 4.

Length of conversion
Specifies the length of the field. For data in INTEL format, this parameter must specify 2 or 4.

User-specified DBCS data
Specifies YES or NO to indicate whether or not shift out/shift in (SO/SI) characters are embedded in the record to indicate the beginning and end of double-byte character data. If YES is specified, and the language is not ideographic-based, CICS/400 converts only the data before the SO character and after the SI character. This parameter is ignored if the type of conversion is not ***CHAR**.

Key conversion data (KEYINF)
Specifies conversion specifications for each field in the key. This parameter is used only for files (**RSRCID(*FILE)**). The key information is found in the OS/400 data description specification (DDS) used to create the file. For each field, this parameter gives:

- Offset from the beginning of the record
- Length
- Type of conversion
- SO/SI indicator

The KEYINF parameter has 4 elements, which define a field and its conversion requirements. You can repeat the 4-element sequence up to 30 times to define up to 30 fields. The 4 elements are specified and processed in exactly the same way as the 4 elements of the CNVINF parameter (see 92).

Selection criteria (SLTCTL)
Selection criteria for selecting different record types within transmitted data, and conversion specifications for records that meet each set of criteria. If this parameter is coded, CICS/400 applies CNVINF specifications to any record types not selected by the SLTCTL criteria.

The SLTCTL parameter has 4 elements, which can be repeated up to 30 times to specify up to 30 selection criteria:

Starting location
Offset in the record where the comparison starts. The default is 0 (comparison begins at the start of the record).

Character or hex format
Format of selection data, character (*CHAR) or hexadecimal (*HEX).

Value to compare against
The selection data that is to be compared against the defined part of the data record. The value is specified in a variable called *text*, which consists of up to 254 hexadecimal digits or 127 alphanumeric characters.

Selection information
Conversion specifications for each field in a record that matches the selection criteria specified in the first 3 elements. For each field, this parameter gives:

- Starting location
- Type of conversion
- Length of conversion
- User specified DBCS data

The selection information has 4 elements which define a field and its conversion requirements. These elements have the same meanings and are specified in the same way as the corresponding elements of the CNVINF and KEYINF parameters. You can define up to 30 fields. **However, unlike the CNVINF and KEYINF parameters, you do not repeat the 4-element sequence – instead you repeat each element separately the required number of times.**

For example, to define three files as follows:

```
Field 1, with elements a1, b1, c1, d1
Field 2, with elements a2, b2, c2, d2
Field 3, with elements a3, b3, c3, d3
```

For CNVINF and KEYINF, code:

```
a1 b1 c1 d1 a2 b2 c2 d2 a3 b3 c3 d3
```

For selection information, code:

```
a1 a2 a3 b1 b2 b3 c1 c2 c3 d1 d2 d3
```

The screen menu enforces these data-entering sequences.

Example

This command adds a CVT entry, called LCVT, to a CICS/400 group, called GROUP, in an OS/400 library called CICSWORK.

```
ADDCICSCVT LIB(CICSWORK) GROUP (GROUP) CMDTYPE(*FILE) RSRCID(REMTFILE)
   CNVCHRID(850 337) CVNINF(6 *CHAR 80 *NO) KEYINF(0 *CHAR 6 *NO)
```

Because no selection criteria are specified, all records for file FILEA are processed using code page 850 and character set 337. The key to be converted consists of character data and starts at the beginning of the record for a length of 6. The rest of the record is character data starting after the key for a length of 80.

This example complements the example on “Example” on page 69, which shows a remote file definition to support function shipping in an application-owning region (AOR). This example shows the corresponding definition in the file-owning region (FOR).

Part 3. Distributed transaction programming

Unlike the other CICS intercommunication facilities, distributed transaction processing requires complex programming. For that reason, the subject is treated here in a separate part, consisting of three chapters of programming guidance.

Apart from the definition of connections to remote systems as described in Chapter 2, “Configuring CICS/400 for intercommunication” on page 11, distributed transaction processing has no special resource-definition requirements. All transactions and programs involved require only local definitions.

Chapter 10. Designing distributed applications

CICS/400 uses the SNA LU6.2 (APPC) protocols for intercommunication. In the design of distributed applications to run under APPC, two major areas are the overall structure of the distributed application and the design of the conversations. There is also a choice of API between the CICS family API and the CPI communications interface, which is available on OS/400 and on some CICS products, but not on CICS/400.

Design objectives

The design of a distributed application involves several conflicting objectives, including performance, ease of maintenance, reliability, security, connectivity, and data integrity.

Avoiding performance problems

If performance is the highest priority, design the application so that data is processed as close to its source as possible. This avoids unnecessary transmission of data across the network. If processing can be deferred, consider batching data locally before transmission.

To maintain performance across an intersystem connection, free a conversation as soon as possible, so that the session can be used by other transactions. In particular, avoid holding a conversation across a terminal wait.

Facilitating maintenance

To correct errors or to adapt to the evolving needs of an organization, distributed applications inevitably need to be modified. Whether these changes are made by the original developers or by others, this task is easier if the distributed design is relatively simple. To facilitate maintenance, consider minimizing the number of transactions involved in a distributed process.

Aiming for reliability

As with ease of maintenance, consider reliability as inversely proportional to the number of transactions in the distributed process—that is, the fewer the transactions, the more reliable the process is likely to be.

Protecting sensitive data

If a distributed application handles security-sensitive data, one way to protect this data is to place it all on a single system. This means that only one of the transactions needs knowledge of how or where the sensitive data is stored.

Maintaining connectivity

If you require connectivity to transactions running in another CICS product, check the *CICS Family: Interproduct Communication* manual to ensure that the required functions are supported between CICS/400 and the remote CICS product.

The following aspects of distributed process design differ from single-system considerations:

Data conversion
If the remote system is a non-EBCDIC APPC logical unit (for example, CICS OS/2), some data conversion may be required either on receipt or on sending of data.

Using multiple conversations
In multiple, serial conversations, different conversation identifiers may be provided to the transaction by CICS. Therefore, do not use a conversation identifier for naming resources, for example, temporary storage queues.

Safeguarding data integrity

OS/400 communication across APPC connections supports synchronization level 2; therefore full synchronization of a distributed unit of work is assured. Confirm flows are also supported.

Designing conversations

When the overall structure of the distributed process has been decided, you can start to design individual conversations. Designing a conversation involves deciding which functions to put into the front-end[16] and back-end [17] transactions, and determining what should be in a distributed unit of work. You must decide how to subdivide the work.

Because a conversation involves transferring data between two transactions, each transaction must know what the other intends. You must consider each front-end and back-end transaction pair as one software unit.

The sequences of commands you can issue on a conversation are governed by a protocol designed to ensure that commands are not issued in inappropriate circumstances. The protocol is based on the concept of a number of conversation states. A conversation state applies only to one side of a conversation and not to the conversation as a whole. In each state, there are a number of commands that can validly be issued. A command, together with its outcome, can change the states on each side of a conversation.

To determine the conversation state, you can use either the STATE parameter on a command or the EXEC CICS EXTRACT ATTRIBUTES STATE command. When a conversation changes state, it is said to have undergone a **state transition**, which generally makes a different set of commands available. The available commands and state transitions are shown in a series of state tables. Which state table you use depends on the sync level and interface chosen. For the state tables for sync levels 0, 1, and 2, see Chapter 13, “State transitions in APPC mapped conversations” on page 145.

Selecting the APPC programming interface

CICS/400 supports the CICS family application programming interface (API) for coding DTP conversations on APPC sessions. Some other members of the CICS family support the Systems Application Architecture* (SAA*) communication interface, which is supported by OS/400, but not by CICS/400.

16. The front-end transaction initiates the conversation.

17. The back-end transaction is initiated by the front-end transaction.

- **CICS API**, is the programming interface of the CICS implementation of the APPC architecture. It consists of EXEC CICS commands, which can be used in COBOL or C application programs.
- **Common Programming Interface Communications** (CPI Communications) is the communications interface defined by SAA. It consists of a set of defined verbs in the form of program calls that are adapted for the language being used.

Table 6 compares CICS API and SAA CPI.

Table 6. CICS/400 API compared with CPI Communications interface

CICS/400 API	SAA CPI communications interface
Supports portability between different members of the CICS family.	Supports portability between systems that support CPI (not CICS/400).
Supports only mapped conversations.	Supports basic conversations programmed in any of the available SAA languages.
Supports outbound PIP data.	Does not support PIP data.
Can be used on the principal facility of a transaction started by automatic transaction initiation (ATI).	Cannot be used on the principal facility of a transaction started by ATI.
Provides commands similar to those used to communicate with IBM 3270 terminals.	Provides commands similar to those used to define the APPC architecture.
Supports passing of parameters on a command.	Requires parameter values to be set by special commands before the issue of the command to which the parameters refer.

For further information about CPI Communications, see the SAA CPI manual referenced below.

It is possible to implement a distributed application in which one partner uses CPI Communications calls and the other uses the CICS API. To do this you must know how the APIs on both sides map to the APPC architecture. This is described in the following two books:

- *CICS/ESA 3.3 Distributed Transaction Programming Guide*, SC33-0783-01
- *CPI Communications Reference*, SC26-4399-09

Chapter 11. APPC mapped conversation flow

General-use programming interface

This section introduces some of the DTP commands for APPC mapped conversations. It introduces each command in the context of a typical conversation flow and ends with a general discussion on how to test the responses from a DTP command.

The main headings in the section are:

Starting the conversation

This section describes how to get a conversation started. The first two subsections explain how the front-end transaction and the back-end transaction initiate the conversation, and the third subsection considers the possibility of failure at the initiation stage. This section also contains program fragments illustrating the use of commands and the checking of response codes.

Conversation initiation

The front-end transaction acquires a session, specifies the conversation characteristics, and requests the startup of the back-end transaction.

Allocating a session to the conversation

Initially, there is no conversation, and therefore no conversation state. By issuing an EXEC CICS ALLOCATE command, the front-end transaction acquires a session to start a new conversation.

The RESP value returned should be checked to ensure that a session has been allocated. If the session is successfully allocated, DFHRESP(NORMAL) is returned, the conversation is in **allocated state** (state 1), and the session identifier (**convid**) in EIBRSRCE must be saved immediately for use in subsequent commands for this conversation. Figure 17 on page 104 shows an example of an EXEC CICS ALLOCATE command.

```
*    ...
 DATA DIVISION.
 WORKING-STORAGE SECTION.
*    ...
 01  FILLER.
     02  WS-CONV         PIC X(4).
     02  WS-RESP         PIC S9(8) BINARY.
     02  WS-STATE        PIC S9(8) BINARY.
     02  WS-SYSID        PIC X(4) VALUE 'SYSB'.
     02  WS-PROC         PIC X(4) VALUE 'BBBB'.
     02  WS-LEN-PROCN    PIC S9(5) BINARY VALUE +4.
     02  WS-SYNC-LVL     PIC S9(5) BINARY VALUE +1.
*    ...

 PROCEDURE DIVISION.
*    ...
     EXEC CICS ALLOCATE SYSID(WS-SYSID) RESP(WS-RESP) END-EXEC.
     IF WS-RESP = DFHRESP(NORMAL)
     THEN MOVE EIBRSRCE TO WS-CONVID
     ELSE
*       ...  No session allocated.  Examine RESP code.
     END-IF.
*    ...
     EXEC CICS CONNECT PROCESS CONVID(WS-CONV) STATE(WS-STATE)
                               RESP(WS-RESP) PROCNAME(WS-PROC)
                               PROCLENGTH(WS-LEN-PROCN)
                               SYNCLEVEL(WS-SYNC-LVL)
     END-EXEC.
     IF WS-RESP = DFHRESP(NORMAL)
     THEN
*       ...  No errors.  Check EIB flags.
     ELSE
*       ...  Conversation not started.  Examine RESP code.
     END-IF.
```

Figure 17. Starting an APPC mapped conversation

Connecting the partner transaction

When the front-end transaction has acquired a session, the next step is to initiate the partner transaction. The state tables show that, in the **allocated state** (state 1), one of the commands available is EXEC CICS CONNECT PROCESS. This command allows the conversation characteristics to be specified and attaches the required back-end transaction. It should be noted that the results of the EXEC CICS CONNECT PROCESS command are placed in the send buffer and are not sent immediately to the partner system. Transmission occurs when the send buffer is cleared, either by sending more data than fits in the send buffer or by issuing an EXEC CICS WAIT CONVID command.

A successful EXEC CICS CONNECT PROCESS command causes the conversation to switch to **send state** (state 2). The program fragment in Figure 17 shows an example of an EXEC CICS CONNECT PROCESS command.

Initial data for the back-end transaction

Various members of the CICS product family support the sending of initial data by the front-end transaction to the back-end transaction. This initial data is called **program initialization parameters**, abbreviated to PIPs or PIP data.

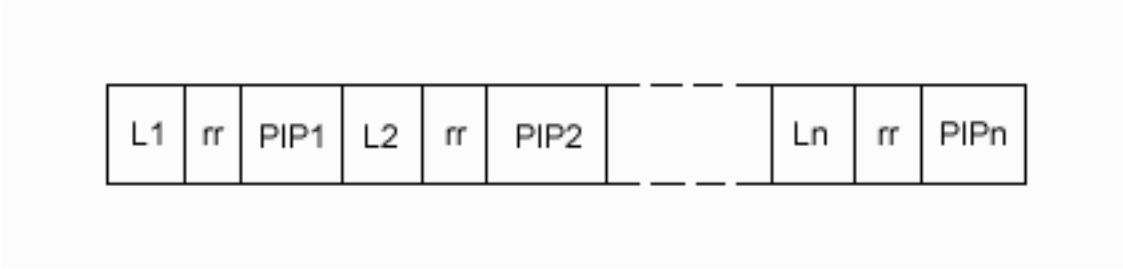
CICS/400 support for PIP data is restricted to the sending of the data.

A CICS/400 back-end transaction cannot receive PIP data. The translator fails if it detects the PIPLIST or PIPLENGTH options on an EXEC CICS EXTRACT PROCESS command. In general, CICS/400 ignores incoming PIP data; however, if the receiving subsystem uses prestarted jobs, and the length of the PIP data is 2000 bytes or more, OS/400 abends the job.

While connecting the back-end transaction, a CICS/400 front-end transaction can place PIPs in specially formatted structures. The PIPLIST option of the EXEC CICS CONNECT PROCESS command is used to send PIPs to the back-end transaction.

PIP data is used only by the two connected transactions and not by the CICS systems. APPC systems other than CICS may not support PIP, or may support it differently.

The PIP data must be formatted into one or more subfields according to the SNA-architected rules. The content of each subfield is defined by the application developer. You should format PIP data as follows:



where Ln is a 2-byte binary integer specifying the length of the subfield, and rr represents 2 reserved bytes. The length includes the length field itself and the length of the reserved field; that is, Ln = (length of PIPn + 4).

CICS inserts information into the reserved fields to make the PIP architecturally correct. The PIPLENGTH option specifies the total length in bytes of the PIP list, and must be in the range 4 through 32763.

Back-end transaction initiation

The back-end transaction is initiated as a result of the front end transaction's EXEC CICS CONNECT PROCESS command. Initially, the back-end transaction should determine the conversation identifier (CONVID). This is not strictly necessary because the session is the back-end transaction's principal facility, making the CONVID parameter nonmandatory for DTP commands on this conversation. However, the CONVID is useful for audit trails. Also, if the back-end transaction is involved in more than one conversation, always specifying the CONVID parameter improves program readability and problem determination.

```
*    ...
 DATA DIVISION.
 WORKING-STORAGE SECTION.
*    ...
 01  FILLER.
     02  WS-CONVID       PIC X(4).
     02  WS-STATE        PIC S9(7) BINARY.
     02  WS-SYSID        PIC X(4) VALUE 'SYSB'.
     02  WS-PROC         PIC X(32).
     02  WS-LEN-PROCN    PIC S9(5) BINARY VALUE +4.
     02  WS-SYNC-LVL     PIC S9(5) BINARY VALUE +1.
*    ...
 01  FILLER.
     02  WS-RECORD       PIC X(100).
     02  WS-MAX-LEN      PIC S9(5) BINARY VALUE +100.
     02  WS-RCVD-LEN     PIC S9(5) BINARY VALUE +0.
*    ...

 PROCEDURE DIVISION.
*    ...
     EXEC CICS ASSIGN FACILITY(WS-CONVID) END-EXEC.
*    ...
*    Extract the conversation characteristics.
*
     EXEC CICS EXTRACT PROCESS PROCNAME(WS-PROC)
                               PROCLENGTH(WS-LEN-PROCN)
                               SYNCLEVEL(WS-SYNC-LVL)
     END-EXEC.
*    ...
*    Receive data from the front-end transaction.
*
     EXEC CICS RECEIVE CONVID(WS-CONVID) STATE(WS-STATE)
                       INTO(WS-RECORD) MAXLENGTH(WS-MAX-LEN)
                       NOTRUNCATE LENGTH(WS-RCVD-LEN)
     END-EXEC.
*
*    ... Check outcome of RECEIVE.
*    ...
```

Figure 18. Starting a back-end APPC mapped transaction at sync level 1

Figure 18 shows a fragment of a back-end transaction that does obtain the conversation identifier. Although the example uses the EXEC CICS ASSIGN command for this purpose, a simpler way would be to access the information in the EIBTRMID field in the EXEC interface block (EIB), which is addressable by CICS application programs.

The back-end transaction can also retrieve its transaction name by issuing the EXEC CICS EXTRACT PROCESS command. In the example shown in Figure 18, CICS places the transaction name in WS-PROC and the length of the name in WS-LEN-PROCN. With the EXEC CICS EXTRACT PROCESS command, the back-end transaction can also retrieve the sync level at which the conversation was started. In the example, CICS places the sync level in WS-SYNC-LVL.

Both the EXEC CICS ASSIGN and the EXEC CICS EXTRACT PROCESS commands are discussed here only to give you some idea of what you can do in the back-end transaction. They are not essential. The back-end transaction starts in **receive state** (state 5), and must issue an EXEC CICS RECEIVE command. By doing this, the back-end transaction receives whatever data the front-end transaction has sent and allows CICS to raise EIB flags and change the conversation state to reflect any request the front-end transaction has issued.

Failure of back-end transaction to start

If the back-end transaction fails to start, CICS raises the TERMERR condition in response to an EXEC CICS command issued by the front-end transaction. Because APPC works asynchronously, TERMERR may not be raised until several commands have been issued. On receipt of a TERMERR condition, EIBERR, EIBFREE, and EIBERRCD are set. For the possible values of EIBERRCD, see Table 10 on page 116.

Before sending data, the front-end transaction can find out whether the back-end transaction has started successfully. One way of doing this is to issue an EXEC CICS SEND CONFIRM command directly after the EXEC CICS CONNECT PROCESS command. This causes the front-end transaction to be suspended until the back-end transaction has responded or has sent the TERMERR condition. The EXEC CICS SEND CONFIRM command is discussed in "How to synchronize a conversation" on page 112.

Transferring data on the conversation

```
*    ...
 DATA DIVISION.
 WORKING-STORAGE SECTION.
*    ...
 01  FILLER.
     02  WS-CONVID       PIC X(4).
     02  WS-STATE        PIC S9(7) BINARY.
*    ...
 01  FILLER.
     02  WS-SEND-AREA    PIC X(70).
     02  WS-SEND-LEN     PIC S9(5) BINARY VALUE +70.
*    ...
 01  FILLER.
     02  WS-RCVD-AREA    PIC X(100).
     02  WS-MAX-LEN      PIC S9(5) BINARY VALUE +100.
     02  WS-RCVD-LEN     PIC S9(5) BINARY VALUE +0.
*    ...

 PROCEDURE DIVISION.
*    ...
     EXEC CICS SEND CONVID(WS-CONVID) STATE(WS-STATE)
                    FROM(WS-SEND-AREA) LENGTH(WS-SEND-LEN)
     END-EXEC.
*    ... Check outcome of SEND.
*    ...
*
     EXEC CICS SEND CONVID(WS-CONVID) STATE(WS-STATE)
                    INVITE WAIT
     END-EXEC.
*    ...
*    Receive data from the partner transaction.
*
     EXEC CICS RECEIVE CONVID(WS-CONVID) STATE(WS-STATE)
                       INTO(WS-RCVD-AREA) MAXLENGTH(WS-MAX-LEN)
                       NOTRUNCATE LENGTH(WS-RCVD-LEN)
     END-EXEC.
*
*    ... Check outcome of RECEIVE.
*    ...
```

Figure 19. Transferring data on a conversation

This section discusses how to pass data between the front- and back-end transactions. The first subsection explains how to send data, the second describes how to switch from sending to receiving data, and the third explains how to receive data. This section also contains a program fragment illustrating the commands described below and the suggested response code checking.

Sending data to the partner transaction

The EXEC CICS SEND command is valid only in **send state** (state 2). Because a successful simple EXEC CICS SEND command leaves the sender in **send state** (state 2), it is possible to issue a number of successive sends. The data from a simple EXEC CICS SEND command is initially stored in a local CICS buffer that is cleared either when it is full or when the transaction requests transmission. Data transmission is deferred to reduce the number of calls to the network. If the partner transaction requires the data to continue processing, the sending transaction can request immediate transmission either by using an EXEC CICS WAIT CONVID command or by using the WAIT option on the EXEC CICS SEND command.

An example of a simple EXEC CICS SEND command can be seen in Figure 19 on page 107.

Switching from sending to receiving data

The column for **send state** (state 2) in the state tables (see 145) shows that there are several ways of switching from **send state** (state 2) to **receive state** (state 5).

One possibility is to use an EXEC CICS RECEIVE command. The state tables show that CICS supplies the INVITE and WAIT options when an EXEC CICS SEND command is followed immediately by an EXEC CICS RECEIVE command.

Another possibility is to use an EXEC CICS SEND INVITE command. The state tables show that after an EXEC CICS SEND INVITE command, the conversation switches to **pendreceive state** (state 3). The column for state 3 shows that an EXEC CICS WAIT CONVID command switches the conversation to **receive state** (state 5).

Still another possibility is to specify the INVITE and WAIT parameters on the EXEC CICS SEND command. The state tables show that after an EXEC CICS SEND INVITE WAIT command, the conversation switches to **receive state** (state 5).

An example of an EXEC CICS SEND INVITE WAIT command can be seen in Figure 19 on page 107. Figure 20 on page 109 illustrates the response-testing sequence after an EXEC CICS SEND INVITE WAIT command with the STATE option. For more information about response testing, see “Checking the outcome of a DTP command” on page 115.

```
*    ...
 DATA DIVISION.
 WORKING-STORAGE SECTION.
*    ...
 01  FILLER.
     02  WS-RESP         PIC S9(7) BINARY.
     02  WS-STATE        PIC S9(7) BINARY.
*    ...

 PROCEDURE DIVISION.
*    ...
* Check return code from SEND INVITE WAIT
     IF WS-RESP = DFHRESP(NORMAL)
     THEN
*       ...  Request successful
        IF EIBERR = LOW-VALUES
        THEN
*          ...  No errors, check state
           IF WS-STATE = DFHVALUE(RECEIVE)
           THEN
*             ...  SEND OK, continue processing
           ELSE
*             ...  Logic error, should never happen
           END-IF
        ELSE
*          ...  Error indicated
           EVALUATE WS-STATE
             WHEN DFHVALUE(RECEIVE)
*                 ...  ISSUE ERROR received, reason in EIBERRCD
             WHEN OTHER
*                 ...  Logic error, should never happen
           END-EVALUATE
        END-IF
     ELSE
*       ...  Examine RESP code for source of error.
     END-IF.
```

Figure 20. Checking the outcome of an EXEC CICS SEND INVITE WAIT command

Receiving data from the partner transaction

```
*    ...
 WORKING-STORAGE SECTION.
*    ...
 01  FILLER.
     02  WS-RESP         PIC S9(8) BINARY.
     02  WS-STATE        PIC S9(8) BINARY.
*    ...
 PROCEDURE DIVISION.
*    ...
* Check return code from RECEIVE
     IF WS-RESP = DFHRESP(EOC)
     OR WS-RESP = DFHRESP(NORMAL)
     THEN
*       ...  Request successful
        IF EIBERR = LOW-VALUES
        THEN
*          ...  No errors, check state
           EVALUATE WS-STATE
             WHEN DFHVALUE(CONFFREE)
*                 ...  Partner issued CONFIRM and LAST
                                    (Sync level 1 only)
             WHEN DFHVALUE(CONFRECEIVE)
*                 ...  Partner issued CONFIRM
                                   (Sync level 1 only)
             WHEN DFHVALUE(CONFSEND)
*                 ...  Partner issued CONFIRM and INVITE
                                   (Sync level 1 only)
             WHEN DFHVALUE(FREE)
*                 ...  Partner issued LAST or FREE
             WHEN DFHVALUE(SEND)
*                 ...  Partner issued INVITE
             WHEN DFHVALUE(RECEIVE)
*                 ...  No state change.  Check EIBCOMPL.
             WHEN OTHER
*              ...  Logic error, should never happen
           END-EVALUATE.
        ELSE
*          ...  Error indicated
           EVALUATE WS-STATE
             WHEN DFHVALUE(RECEIVE)
*                 ...  ISSUE ERROR received, reason in EIBERRCD
             WHEN DFHVALUE(FREE)
*                 ...  ISSUE ABEND received, reason in EIBERRCD
             WHEN OTHER
*                 ...  Logic error, should never happen
           END-EVALUATE
        END-IF
     ELSE
*       ...  Examine RESP code for source of error
     END-IF.
```

Figure 21. Checking the outcome of an EXEC CICS RECEIVE command

The EXEC CICS RECEIVE command is used to receive data from the connected partner. The rows in the state tables for the EXEC CICS RECEIVE command show the EIB fields that should be tested after issuing an EXEC CICS RECEIVE command. As well as showing which field should be tested, the state tables also show the order in which the tests should be made.

As an alternative to testing the EIB fields it is possible to test the resulting conversation state; this is shown in Figure 21. The conversation state can be

meaningfully tested only after issuing a command with the STATE parameter or by using the EXEC CICS EXTRACT ATTRIBUTES STATE command. Note that the RESP value returned and EIBERR should always be tested.

If EIBNODAT is set on (X'FF'), no data has been received. For more information about response testing, see "Checking the outcome of a DTP command" on page 115. For information about testing the conversation state, see "Testing the conversation state" on page 154.

An example of an EXEC CICS RECEIVE command with the STATE parameter can be seen in Figure 19 on page 107. Figure 21 illustrates the response-testing and state-testing sequence.

The EXEC CICS CONVERSE command

The EXEC CICS CONVERSE command combines the functions of the EXEC CICS SEND INVITE WAIT and EXEC CICS RECEIVE commands, and is useful when a transaction needs a response from its partner transaction to continue processing.

Communicating errors across a conversation

The APPC mapped API provides commands to enable transactions to pass error notification across a conversation. There are three commands depending on the severity of the error—the most severe, EXEC CICS ISSUE ABEND, causes the conversation to terminate abnormally and is described in "Emergency termination of a conversation" on page 114. The other two commands are described below.

Requesting INVITE from the partner transaction

If a transaction is receiving data on a conversation and wants to send, it can use the EXEC CICS ISSUE SIGNAL command to ask the partner transaction to issue an EXEC CICS SEND INVITE command. When the EXEC CICS ISSUE SIGNAL request is received, EIBSIG=X'FF' and the SIGNAL condition is raised. Note that on receipt of an EXEC CICS ISSUE SIGNAL command, a transaction is **not** obliged to issue an EXEC CICS SEND INVITE command.

Demanding INVITE from the partner transaction

If a transaction needs to send an immediate error notification to the partner transaction, it can use the EXEC CICS ISSUE ERROR command. This command is also one of the preferred negative responses to an EXEC CICS SEND CONFIRM command. When the EXEC CICS ISSUE ERROR command is received, EIBERR=X'FF' and the first two bytes of EIBERRCD are X'0889'. This error condition cannot be processed by an EXEC CICS HANDLE CONDITION command and cannot be tested by the RESP parameter that can be coded on any command. It can be detected only by testing EIBERR and EIBERRCD.

If an EXEC CICS ISSUE ERROR command is used in **receive state** (state 5), all incoming data is purged until an EXEC CICS SEND INVITE or EXEC CICS SEND LAST command is received. If an EXEC CICS SEND LAST command is received, no error indication is sent to the partner transaction, EIBFREE=X'FF' and the conversation is switched to **free state** (state 12).

If an EXEC CICS SEND INVITE command is received, the conversation is switched to **send state** (state 2). It is normal programming practice to communicate the

reason for the EXEC CICS ISSUE ERROR command to the partner transaction. The EXEC CICS CONVERSE command could be used to send an appropriate error message and receive a reply.

Because an EXEC CICS ISSUE ERROR command is allowed in both **send state** (state 2) and **receive state** (state 5), it is possible for both communicating transactions to use an EXEC CICS ISSUE ERROR command at the same time. When this occurs, only one of the EXEC CICS ISSUE ERROR commands is effective. The other is purged with incoming data. However, both EXEC CICS ISSUE ERROR commands will appear to have completed successfully and the transaction whose EXEC CICS ISSUE ERROR command was purged will pick up EIBERR=X'FF' on a subsequent command.

Safeguarding data integrity (using sync level 1)

CICS/400 supports sync-level 2 protocols (using the SYNCPOINT and SYNCPOINT ROLLBACK commands) which are described in Chapter 12, "Syncpointing a distributed process" on page 121. This section describes what you can do to safeguard data integrity across connected transactions that can use only sync-level 1 protocols. For sync-level 1 conversations, you have two CICS synchronization commands available:

```
EXEC CICS SEND CONFIRM
EXEC CICS ISSUE CONFIRMATION
```

How to synchronize a conversation

A confirmation exchange affects a single specified conversation and involves only the two commands mentioned above:

1. The conversation that is in **send state** (state 2) issues an EXEC CICS SEND CONFIRM command causing a request for confirmation to be sent to the partner transaction. The transaction suspends awaiting a response.
2. The partner transaction receives a request for confirmation. It can then respond positively by issuing an EXEC CICS ISSUE CONFIRMATION command. Alternatively, it can respond negatively by using the EXEC CICS ISSUE ERROR or EXEC CICS ISSUE ABEND commands.

The following sections describe these commands in more detail. The descriptions refer to the state tables.

Requesting confirmation

The CONFIRM parameter of the EXEC CICS SEND command clears the conversation send buffer; that is, it causes a transmission to occur. When the conversation is in **send state** (state 2), you can send data with the EXEC CICS SEND CONFIRM command. You can also specify either the INVITE or the LAST parameter.

The **send state** (state 2) column of the state table for APPC mapped conversations at sync level 1 148 shows what happens for the possible combinations of the CONFIRM, INVITE, and LAST parameters. After an EXEC CICS SEND CONFIRM command, without the INVITE or LAST parameters, the conversation remains in **send state** (state 2). If the INVITE parameter is used, the conversation switches to **receive state** (state 5). If the LAST parameter is used, the conversation switches to **free state** (state 12).

A similar effect to an EXEC CICS SEND LAST CONFIRM command can be achieved by using the command sequence:

```
EXEC CICS SEND LAST
EXEC CICS SEND CONFIRM
```

Note from the state tables that the EXEC CICS SEND LAST command puts the conversation into **pendfree state** (state 4), so data cannot be sent with an EXEC CICS SEND CONFIRM command used in this way.

The form of command used depends on how the conversation is to continue if the required confirmation is received. However, the response from an EXEC CICS SEND CONFIRM command **must** always be checked. See Checking the response to EXEC CICS SEND CONFIRM commands.

Receiving and replying to a confirmation request

On receipt of a confirmation request, the EIB and conversation state will be set depending on the request issued by the partner transaction. These together with the contents of the EIBCONF, EIBRECV, and EIBFREE fields are shown in Table 7.

Table 7. Indications of a confirmation request

Command issued by partner transaction	On receipt of request			
	Conversation state	EIBCONF	EIBRECV	EIBFREE
SEND CONFIRM	confreceive (state 6)	X'FF'	X'FF'	X'00'
SEND INVITE CONFIRM	confsend (state 7)	X'FF'	X'00'	X'00'
SEND LAST CONFIRM	conffree (state 8)	X'FF'	X'00'	X'FF'

There are three ways of replying:

1. Reply positively with an EXEC CICS ISSUE CONFIRMATION command.
2. Reply negatively with an EXEC CICS ISSUE ERROR command. This reply puts the conversation into **send state** (state 2) regardless of the partner transaction request.
3. Abnormally end the conversation with an EXEC CICS ISSUE ABEND command. This makes the conversation unusable and an EXEC CICS FREE command must be issued immediately.

Checking the response to EXEC CICS SEND CONFIRM commands

After issuing EXEC CICS SEND [INVITE³LAST] CONFIRM, it is important to test EIBERR to determine the partner's response. Table 8 shows how the partner's response is indicated by EIB flags and the conversation states.

Table 8. Indications of responses to EXEC CICS SEND CONFIRM

Command issued in reply by partner transaction	On receipt of response		
	Conversation state	EIBERR	EIBFREE
ISSUE CONFIRMATION	dependent on original SEND [INVITE|LAST] CONFIRM request	X'00'	X'00'
ISSUE ERROR	receive (state 5)	X'FF'	X'00'
ISSUE ABEND	free (state 12)	X'FF'	X'FF'

If EIBERR=X'00', the partner has replied with an EXEC CICS ISSUE CONFIRMATION command.

If the partner replies with an EXEC CICS ISSUE ERROR command, this is indicated by EIBERR=X'FF' and the first two bytes of EIBERRCD = X'0889'. When the partner sends an EXEC CICS ISSUE ERROR command in response to an EXEC CICS SEND LAST CONFIRM command, the LAST parameter is ignored and the conversation is **not** terminated. The conversation state is switched to **receive state** (state 5).

If the partner replies with an EXEC CICS ISSUE ABEND command, the TERMERR condition is raised. In addition, EIBERR and EIBFREE are set, and the first two bytes of EIBERRCD=X'0864'. The conversation is switched to **free state** (state 12).

Ending the conversation

The following sections describe the different ways a conversation can end, either unexpectedly or under transaction control. To end a transaction, one transaction issues a request for termination and the other receives this request. When this has happened the conversation is unusable and **both** transactions must issue an EXEC CICS FREE command to release the session.

Normal termination of a conversation

The EXEC CICS SEND LAST command is used to terminate a conversation. It should be used in conjunction with the WAIT or CONFIRM options, or the EXEC CICS WAIT CONVID command. This is described in Table 9.

Table 9. Command sequences for ending a conversation

Sync level	Command sequence
0	SEND LAST WAIT FREE
1	SEND LAST CONFIRM FREE
2	SEND LAST SYNCPOINT FREE

Note: CICS/400 does not support the CICS/ESA LU6.2 deviation that allows a sync-level 2 conversation to be ended by a FREE command when in **send state** (state 2), nor does CICS/400 permit the use of SEND LAST WAIT or SEND LAST CONFIRMATION over sync-level 2 conversations. CICS/400 generates an ATCU abend if any of these are attempted.

Emergency termination of a conversation

The EXEC CICS ISSUE ABEND command provides a means of abnormally ending the conversation.

The EXEC CICS ISSUE ABEND command can be issued by either transaction, irrespective of whether it is in send or receive state, at any time after the conversation has started. For a conversation in **send state** (state 2), any deferred data that is waiting for transmission is cleared before the EXEC CICS ISSUE ABEND command is transmitted.

The transaction that issues the EXEC CICS ISSUE ABEND command is not itself abended. It must, however, issue an EXEC CICS FREE command for the conversation unless it is designed to terminate immediately.

If an EXEC CICS ISSUE ABEND command is issued in **receive state** (state 5), CICS purges all incoming data until an INVITE, syncpoint request, or LAST indicator is received. If LAST is received, no abend indication is sent to the partner transaction.

If an EXEC CICS ISSUE ABEND command is received, CICS raises the TERMERR condition, sets on EIBERR (=X'FF') and EIBFREE (=X'FF'), and places X'0864' in the first two bytes of EIBERRCD. The only command that can be subsequently issued for the conversation is EXEC CICS FREE.

Unexpected termination of a conversation

If any of the following occur during a DTP conversation, the conversation is terminated abnormally and the TERMERR condition is raised on the next command that accesses the conversation:

Partner fails.

Partner terminates abnormally under application control.

Session goes out of service.

In addition EIBERR and EIBFREE are set on (X'FF') and EIBERRCD contains a sense code representing the reason for the error. Possible sense codes include:

X'1008600B' - session has failed due to a protocol error

X'A0000100' - temporary session failure

X'A0010100' - RTIMEOUT triggered

Checking the outcome of a DTP command

Checking the response from a DTP command can be separated into three stages:

1. Testing for request failure
2. Testing for indicators received on the conversation
3. Testing the conversation state

Testing for request failure

Testing for request failure is the same as for other EXEC CICS commands in that conditions are raised and can be handled using HANDLE CONDITION or RESP. EIBRCODE will also contain an error code. Note that when an ISSUE ABEND has been received, and it is to be handled, a HANDLE ABEND should be used rather than a HANDLE CONDITION.

Testing for indicators

If the request has not failed, it is then possible to test for indicators received on the conversation. These are returned to the application in the EIB. The following EIB fields are relevant to all DTP commands:

EIBERR

when set to X'FF' indicates an error has occurred on the conversation. The reason is in EIBERRCD. This could be as a result of an ISSUE ERROR, ISSUE ABEND, or SYNCPOINT ROLLBACK command issued by the partner transaction. EIBERR can be set as a result of any command that can be issued while the conversation is in **receive state** (state 5) or following any command that causes a transmission to the partner system. It is safest to test EIBERR in conjunction with EIBFREE and EIBSYNRB after every DTP command.

EIBERRCD
contains the error code associated with EIBERR. If EIBERR is not set, this field is not used.

EIBFREE
when set to X'FF' indicates that the partner transaction had ended the conversation. It should be tested along with EIBERR and EIBSYNC to find out exactly how to end the conversation.

EIBSIG
when set to X'FF' indicates the partner transaction or system has issued an ISSUE SIGNAL command.

EIBSYNRB
when set to X'FF' indicates the partner transaction or system has issued a SYNCPOINT ROLLBACK command. (This is relevant only for conversations at sync level 2.)

Table 10 shows how these EIB fields interact.

Table 10. Interaction between some EIB fields—all DTP commands

EIB- ERR	EIB- FREE	EIB- SYNRB	EIB- ERRCD	Description
X'FF'	X'00'	X'00'	X'08890000' X'08890001'	The partner transaction has sent ISSUE ERROR
X'FF'	X'00'	X'00'	X'08890100' X'08890101'	The partner system has sent ISSUE ERROR
X'FF'	X'FF'	X'00'	X'08640000'	The partner transaction has sent ISSUE ABEND
X'FF'	X'FF'	X'00'	X'08640001'	The partner system has sent ISSUE ABEND
X'FF'	X'FF'	X'00'	X'08640002'	A partner resource has timed out
X'FF'	X'FF'	X'00'	X'1008600B'	The session has failed due to a protocol error
X'FF'	X'FF'	X'00'	X'A0000100'	A temporary session failure
X'FF'	X'FF'	X'00'	X'A0010100'	RTIMOUT has been triggered. (The task has timed out while waiting for terminal input.)
X'FF'	X'FF'	X'00'	X'10086032'	The PIP data sent with the CONNECT PROCESS was incorrectly specified
X'FF'	X'FF'	X'00'	X'10086034'	The partner system does not support mapped conversations
X'FF'	X'FF'	X'00'	X'080F6051'	The partner transaction failed security check
X'FF'	X'FF'	X'00'	X'10086041'	The partner transaction does not support the sync level requested on the CONNECT PROCESS
X'FF'	X'FF'	X'00'	X'10086021'	The partner transactions name is not recognized by the partner system
X'FF'	X'FF'	X'00'	X'084C0000'	The partner system cannot start the partner transaction
X'FF'	X'FF'	X'00'	X'084B6031'	The partner system is temporarily unable to start the partner transaction
X'FF'	X'00'	X'FF'	X'08240000'	The partner transaction or system has issued SYNCPOINT ROLLBACK
X'00'	X'00'	—	—	The command completed successfully

In addition the following EIB fields are relevant only to the RECEIVE and CONVERSE commands:

EIBCOMPL
when set to X'FF' indicates that all the data sent at one time has been received. This field is used in conjunction with the RECEIVE NOTRUNCATE command.

EIBCONF
when set to X'FF' indicates that the partner transaction has issued a SEND CONFIRM command and requires a response.

EIBEOC
when set to X'FF' indicates that an end-of-chain indicator has been received. This field is normally associated with a successful RECEIVE command.

EIBNODAT
when set to X'FF' indicates that no application data has been received.

EIBRECV
is only used when EIBERR is not set. When EIBRECV is on (X'FF'), another RECEIVE is required.

EIBSYNC
when set to X'FF' indicates that the partner transaction or system has requested a syncpoint. (This is relevant only for conversations at sync level 2.)

Table 11 shows how some of these EIB fields interact for RECEIVE and CONVERSE commands.

Table 11. Interaction between some EIB fields—RECEIVE and CONVERSE commands only

EIB- ERR	EIB- FREE	EIB- RECV	EIB- SYNC	EIB- CONF	Description
X'00'	X'00'	X'00'	X'00'	X'00'	The partner transaction or system has issued SEND INVITE WAIT. The local program is now in send state.
X'00'	X'00'	X'00'	X'FF'	X'00'	The partner transaction or system has issued SEND INVITE, followed by a SYNCPOINT. The local program is now in syncsend state.
X'00'	X'00'	X'00'	X'00'	X'FF'	The partner transaction or system has issued SEND INVITE CONFIRM. The local program is now in confsend state.
X'00'	X'00'	X'FF'	X'00'	X'00'	The partner transaction or system has issued SEND or SEND WAIT. The local program is in receive state.
X'00'	X'00'	X'FF'	X'FF'	X'00'	The partner transaction or system has issued a SYNCPOINT. The local program is in syncreceive state.
X'00'	X'00'	X'FF'	X'00'	X'FF'	The partner transaction or system has issued a SEND CONFIRM. The local program is in confreceive state.
X'00'	X'FF'	X'00'	X'00'	X'00'	The partner transaction or system has issued a SEND LAST WAIT. The local program is in free state.
X'00'	X'FF'	X'00'	X'FF'	X'00'	The partner transaction or system has issued a SEND LAST followed by a SYNCPOINT. The local program is in syncfree state.
X'00'	X'FF'	X'00'	X'00'	X'FF'	The partner transaction or system has issued a SEND LAST CONFIRM. The local program is in conffree state.

After analyzing the EIB fields, you can test the conversation state to determine which DTP commands you can issue next. See Chapter 13, “State transitions in APPC mapped conversations” on page 145.

Checking EIB fields and the conversation state

Most of the information supplied by EIB indicator fields can also be obtained from the conversation state. Although the conversation state is easier to test, you cannot ignore EIBERR (and EIBERRCD).

For example, if after a SEND INVITE WAIT or a RECEIVE command has been issued, the conversation is in **receive state** (state 5), only EIBERR indicates that the partner transaction has sent an ISSUE ERROR. This is illustrated in Figure 20 on page 109 and Figure 21 on page 110.

It should be noted that the state tables provided contain not only states and commands issued, but also relevant EIB field settings. The order in which these EIB fields are shown provides a sensible sequence of checks for an application.

Summary of CICS commands for APPC mapped conversations

Table 12 shows the CICS commands used in APPC mapped conversations.

Table 12. Summary of CICS commands used in mapped conversations

Use to ...	Sync levels	CICS Command	Page
Acquire a session.	0,1,2	ALLOCATE	103
Initiate a conversation.	0,1,2	CONNECT PROCESS	104
Access session-related information.	0,1,2	EXTRACT PROCESS	105
Send data and control information to the conversation partner.	0,1,2	SEND	108
Receive data from the conversation partner.	0,1,2	RECEIVE	110
Send and receive data on the conversation.	0,1,2	CONVERSE	111
Transmit any deferred data or control indicators.	0,1,2	WAIT CONVID	108
Reply positively to SEND CONFIRM.	1,2	ISSUE CONFIRMATION	113
Prepare for sync point processing.	2	ISSUE PREPARE	122
Inform the conversation partner of a program-detected error.	0,1,2	ISSUE ERROR	111
Signal an unusual condition to the conversation partner, usually against the flow of data.	0,1,2	ISSUE SIGNAL	111
Inform the conversation partner that the conversation should be abandoned.	0,1,2	ISSUE ABEND	114
Free the session.	0,1,2	FREE	114
Request all changes to recoverable resources be committed. The failure of any single resource to commit will result in all of the resources being backed out.	2 (See note)	SYNCPOINT	121

Table 12. Summary of CICS commands used in mapped conversations (continued)

Use to ...	Sync levels	CICS Command	Page
Request that any changes to recoverable resources since the last syncpoint be backed out.	2 (See note)	SYNCPOINT ROLLBACK	122
Note: SYNCPOINT or SYNCPOINT ROLLBACK may be issued by a transaction that has sync level 0 or sync level 1 conversations. In such cases, nothing is propagated over these conversations. Only local resources (together with any resources managed over a sync level 2 conversation) will be syncpointed.			

For definitive programming interface information about CICS/400 commands, see the *CICS for iSeries Application Programming Guide*.

End of General-use programming interface

Chapter 12. Syncpointing a distributed process

This chapter discusses how to include syncpointing in a distributed process. The chapter concentrates on the programming aspects of using the EXEC CICS SYNCPOINT [ROLLBACK][18] command across APPC mapped conversations at sync level 2. This includes issuing syncpoint requests and receiving them, because they are transmitted to all partners connected on conversations at sync level 2. The chapter also describes how these partners are given the opportunity to back out even though they have been requested to commit.

The EXEC CICS SYNCPOINT command

The EXEC CICS SYNCPOINT command is used to commit recoverable resources. In a distributed transaction processing (DTP) environment, the effect of the SYNCPOINT command is propagated across all conversations using sync level 2. So, no matter how many DTP transactions are connected by conversations at sync level 2, the distributed process should be designed so that only one of the transactions initiates syncpoint activity for the distributed unit of work. When issuing the SYNCPOINT command, this transaction, known as the **syncpoint initiator** must be in **send state** (state 2), **pendreceive state** (state 3), or **pendfree state** (state 4) on all its conversations at sync level 2. Any transaction that receives the syncpoint request becomes a **syncpoint agent**.

A syncpoint agent is in **receive state** on its conversation with the syncpoint initiator and becomes aware of the syncpoint request by testing EIBSYNC after issuing an EXEC CICS RECEIVE command. If it decides to respond positively by issuing SYNCPOINT, it must be in an appropriate state on all the conversations with its own agents, for which it has become syncpoint initiator. If an agent transaction responds negatively to a syncpoint request by issuing SYNCPOINT ROLLBACK, the initiator sees EIBRLDBK set (to X'FF'), and this must be tested for on return from the SYNCPOINT command.

Your transaction design should ensure that all participating transactions are in the correct conversation state before a SYNCPOINT command is issued.

When a syncpoint agent receives the syncpoint request, it is given the opportunity to respond positively (to commit recoverable resources) with a SYNCPOINT command or negatively (to back out recoverable resources) with a SYNCPOINT ROLLBACK command. For information on backing out recoverable resources, see “The EXEC CICS SYNCPOINT ROLLBACK command” on page 122.

Examples of these commands are given in “Synchronizing two CICS systems” on page 123 and “Synchronizing three or more CICS systems” on page 136.

18. The SAA equivalents for this syncpointing command (SRRCMIT and SRRBACK) are described in the *SAA Common Programming Interface (CPI) Resource Recovery Reference*, SC31-6821-01.

The EXEC CICS ISSUE PREPARE command

The EXEC CICS ISSUE PREPARE command is used to send the initial syncpoint flow to a selected partner on an APPC conversation at sync level 2. Depending on the partner's response, this command can then be followed by a SYNCPOINT or SYNCPOINT ROLLBACK command.

The reasons for using ISSUE PREPARE are as follows:

1. In complex DTP involving several conversing transactions, an ISSUE ERROR command from one of the transactions may not reach the syncpoint initiator in time to prevent it from issuing a SYNCPOINT command. This can lead to complex backout procedures for the distributed unit of work.

   Use ISSUE PREPARE as a way of flushing any error responses from the network.
2. If one or more syncpoint agents are not completely "reliable", use ISSUE PREPARE to check the status of these agents before proceeding with a general distributed syncpoint. Receiving ISSUE PREPARE is exactly the same as receiving SYNCPOINT. The partner program cannot detect any difference.

The EXEC CICS SYNCPOINT ROLLBACK command

The EXEC CICS SYNCPOINT ROLLBACK command is used to back out changes to recoverable resources. In a DTP environment, the effect of the SYNCPOINT command is propagated across all conversations using sync level 2. A SYNCPOINT ROLLBACK command can be issued in any conversation state. If the command is issued when a conversation is in **receive state** (state 5), incoming data on that conversation is purged as described for the ISSUE ERROR and ISSUE ABEND commands.

When a transaction receives a SYNCPOINT ROLLBACK in response to a syncpoint request, the EIBRLDBK indicator is set. If SYNCPOINT ROLLBACK is received in response to any other request, the EIBERR and EIBSYNRB indicators are set.

Conversation state after SYNCPOINT ROLLBACK

(Note that this section also applies when the ROLLEDBACK condition is returned from an EXEC CICS SYNCPOINT command.)

Considerations in determining the state after SYNCPOINT ROLLBACK depend on the CICS family type and release of the partner system:

- In most cases, the conversation state of each partner is restored to the state at the beginning of the distributed unit of work.
- In CICS/ESA versions before Version 3.2.1, the rollback initiator completes backout processing in **send state** (state 2), and the partner completes in **receive state** (state 5).
- If a session failure or notification of a deallocate abend occurs during SYNCPOINT ROLLBACK processing, the command still completes successfully. If the same thing happens during SYNCPOINT processing, the command may complete successfully with EIBRLDBK set. In such circumstances, the conversation on which the failure or abend occurred will be in **free state** (state 12).

To avoid potential state problems, you should check the conversation state by using the STATE option on the next APPC command. To avoid potential abends

altogether, you are recommended to follow all EXEC CICS SYNCPOINT ROLLBACK commands with an EXEC CICS EXTRACT ATTRIBUTES STATE command.

When a backout is required

A backout is required in the following circumstances:

- When SYNCPOINT ROLLBACK is received
- After ISSUE ABEND is sent
- After EIBERR and EIBFREE are returned together

The conversation state does not always reflect the requirement to back out. However, CICS is aware of this requirement and converts the next SYNCPOINT request to a SYNCPOINT ROLLBACK request. If no SYNCPOINT or SYNCPOINT ROLLBACK request is issued before the end of the task, the task is abended with code ASPN, and all recoverable resources are backed out.

Synchronizing two CICS systems

This section gives examples of how to commit and back out changes to recoverable resources made by two DTP transactions connected on a conversation using sync level 2.

The examples illustrate the following CICS command sequences and session failure scenarios:

- SYNCPOINT in response to SYNCPOINT 123
- SYNCPOINT in response to ISSUE PREPARE 125
- SYNCPOINT ROLLBACK in response to SYNCPOINT ROLLBACK 126
- SYNCPOINT ROLLBACK in response to SYNCPOINT 127
- SYNCPOINT ROLLBACK in response to ISSUE PREPARE 128
- ISSUE ERROR in response to SYNCPOINT 129
- ISSUE ERROR in response to ISSUE PREPARE 130
- ISSUE ABEND in response to SYNCPOINT 131
- ISSUE ABEND in response to ISSUE PREPARE 132
- Session failure in response to SYNCPOINT 133
- Session failure in response to ISSUE PREPARE 135
- Session failure in response to SYNCPOINT ROLLBACK 135

EXEC CICS SYNCPOINT in response to EXEC CICS SYNCPOINT

Figure 22, Figure 23, and Figure 24 on page 125 illustrate the effect of SEND, SEND INVITE, or SEND LAST preceding SYNCPOINT on an APPC mapped conversation. The figures also show the conversation state before each command and the state and EIB fields set after each command.

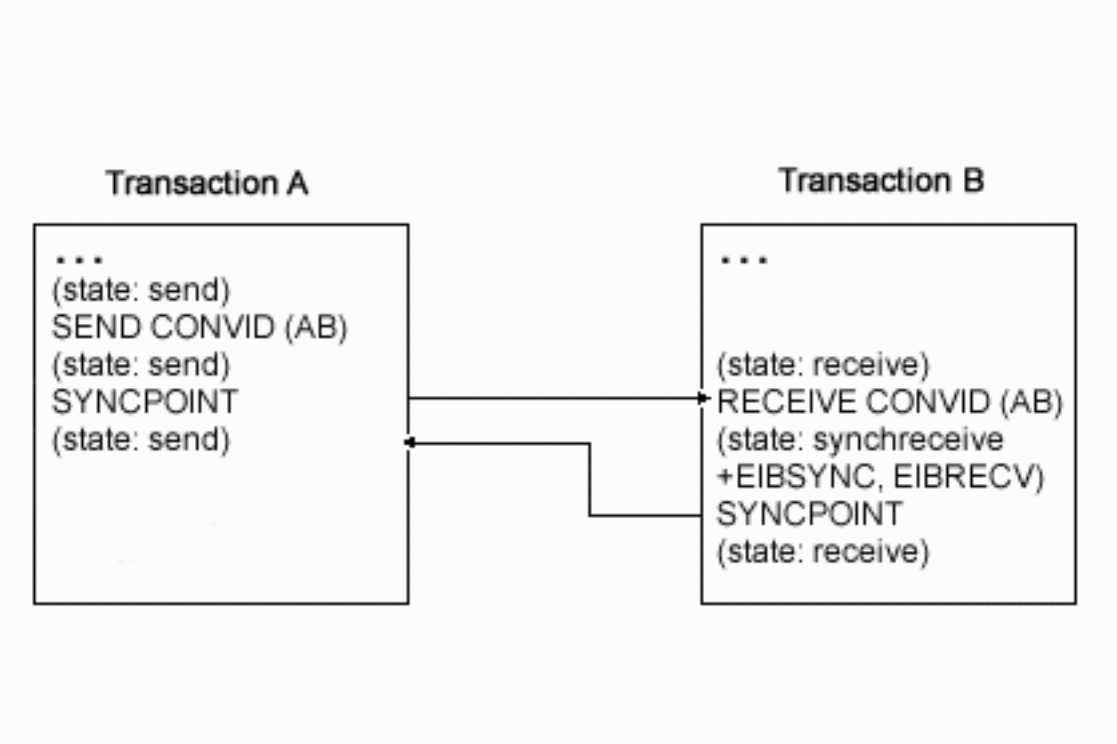


Figure 22. SYNCPOINT in response to SEND followed by SYNCPOINT on an APPC mapped conversation

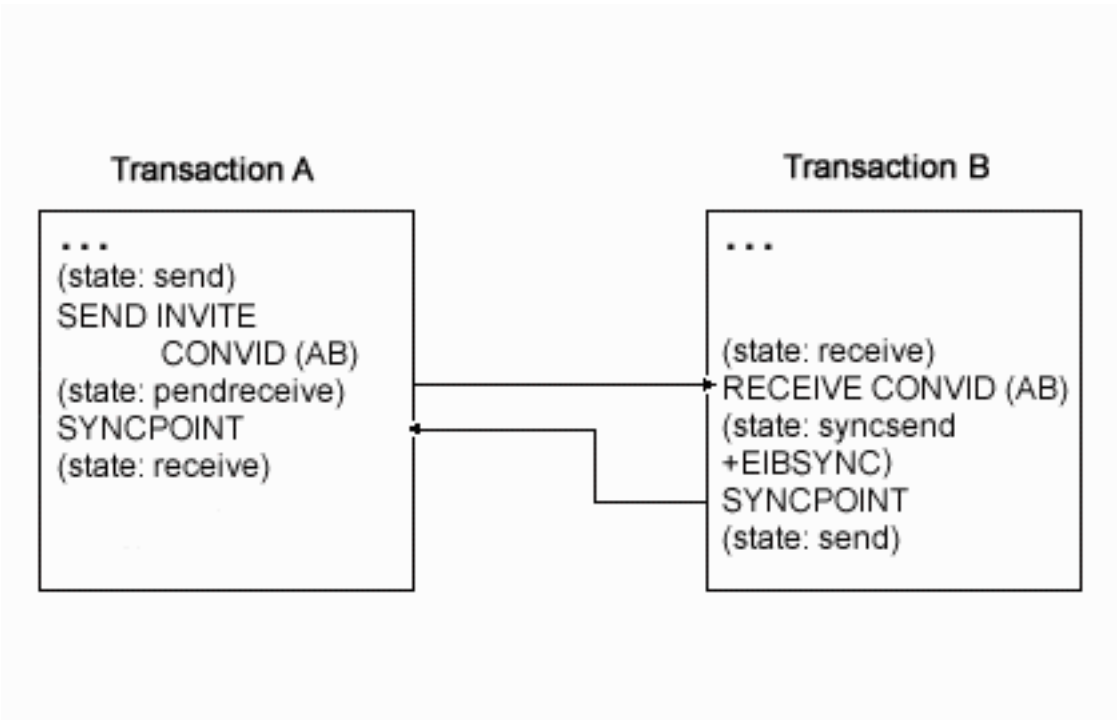


Figure 23. SYNCPOINT in response to SEND INVITE followed by SYNCPOINT on an APPC mapped conversation

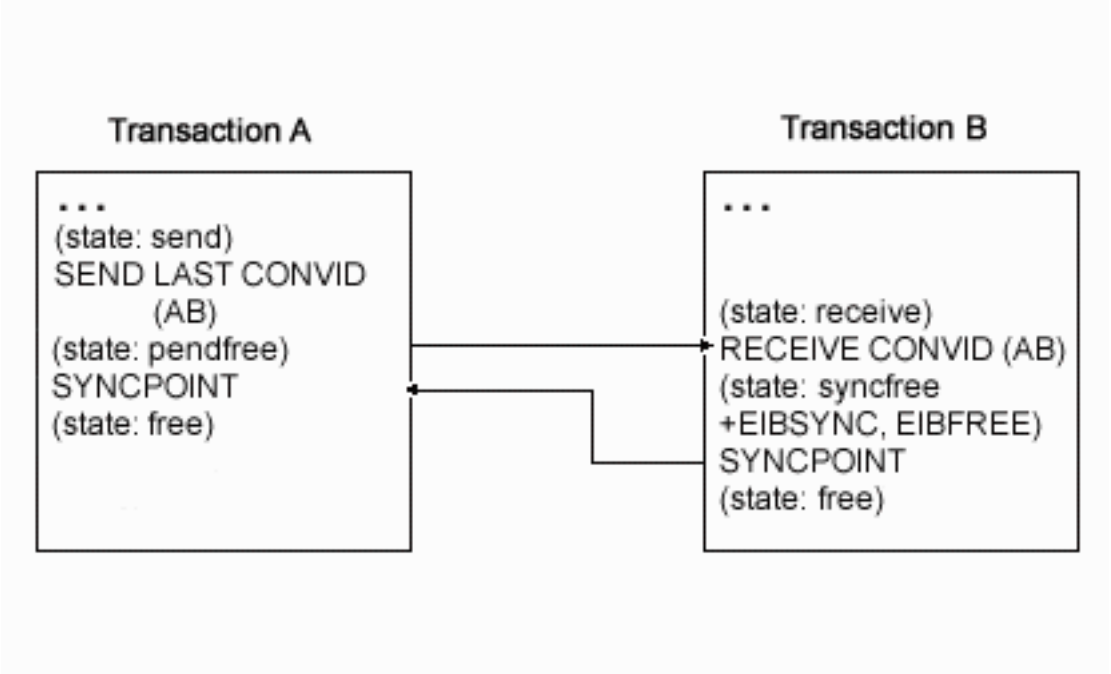


Figure 24. SYNCPOINT in response to SEND LAST followed by SYNCPOINT on an APPC mapped conversation

EXEC CICS SYNCPOINT in response to EXEC CICS ISSUE PREPARE

Figure 25 on page 126 illustrates a SYNCPOINT command being used in response to ISSUE PREPARE on an APPC mapped conversation. The figure also shows the conversation state before each command and the state and EIB fields set after each command.

Note that it is also possible to use an ISSUE PREPARE command in **pendreceive state** (state 3) and **pendfree state** (state 4).

Note also that, although the ISSUE PREPARE command in Figure 25 on page 126 returns with the conversation in **syncsend state** (state 10), the only commands available for use on that conversation are SYNCPOINT and SYNCPOINT ROLLBACK. All other commands abend with code ATCV.

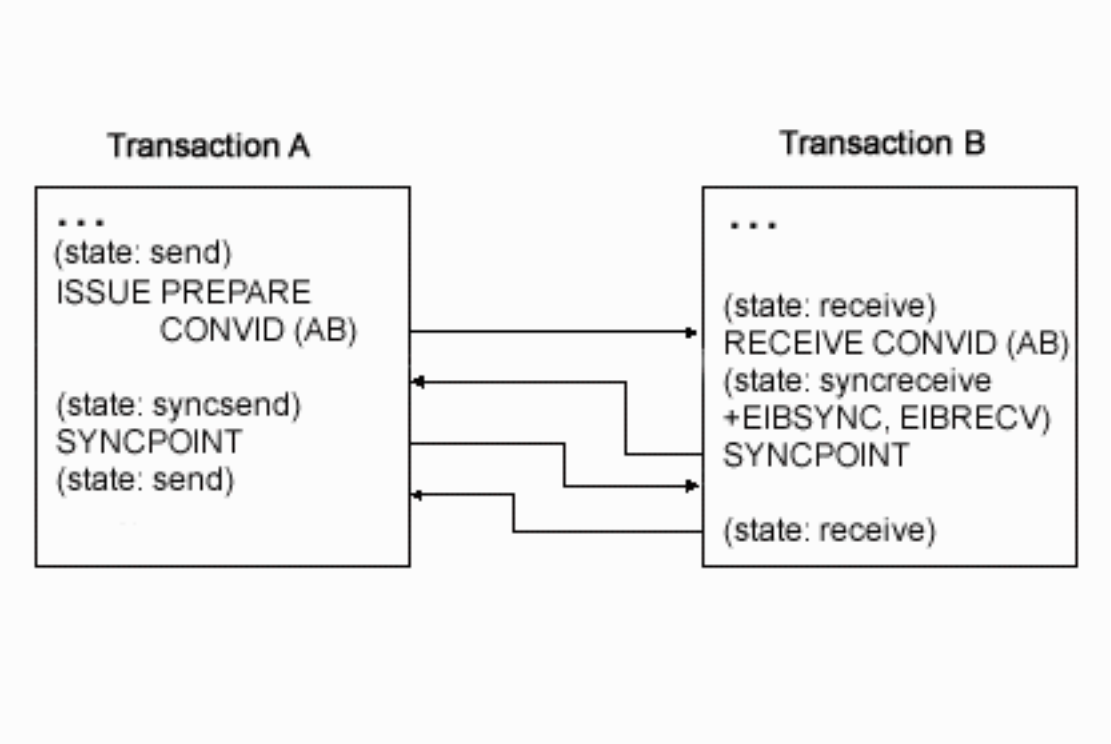


Figure 25. SYNCPOINT in response to ISSUE PREPARE on an APPC mapped conversation

EXEC CICS SYNCPOINT ROLLBACK in response to EXEC CICS SYNCPOINT ROLLBACK

Figure 26 on page 127 illustrates a SYNCPOINT ROLLBACK command being used in response to SYNCPOINT ROLLBACK on an APPC mapped conversation. The figure also shows the conversation state before each command and the state and EIB fields set after each command.

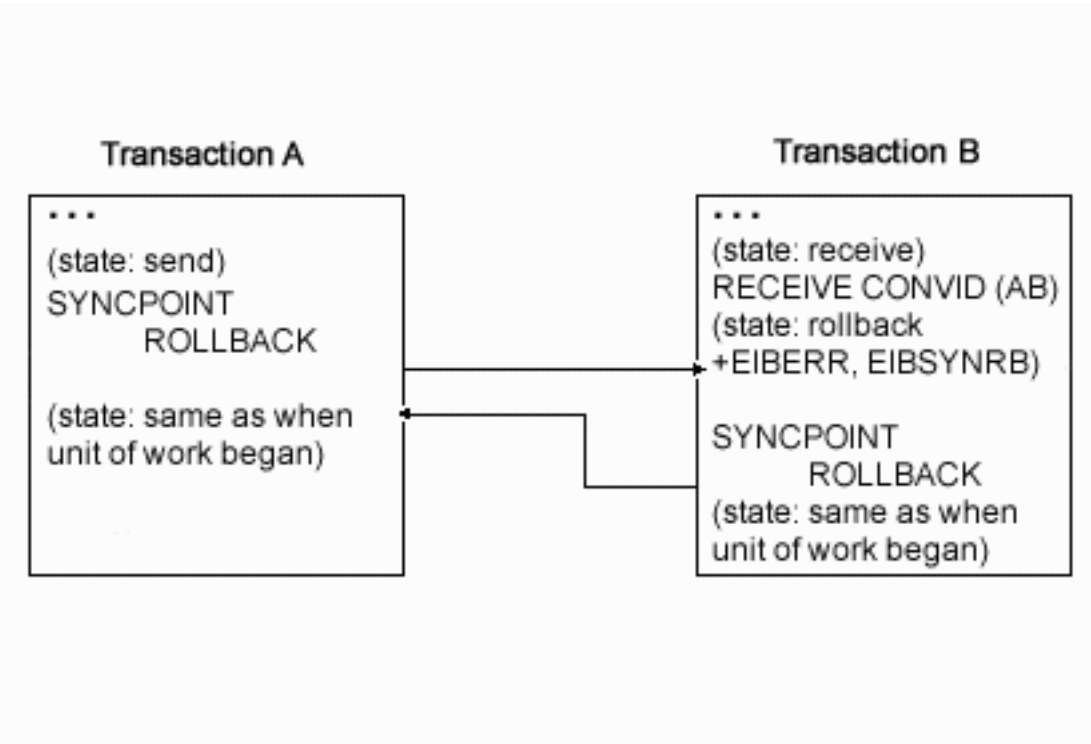


Figure 26. SYNCPOINT ROLLBACK in response to SYNCPOINT ROLLBACK on an APPC mapped conversation

EXEC CICS SYNCPOINT ROLLBACK in response to EXEC CICS SYNCPOINT

Figure 27 on page 128 illustrates a SYNCPOINT ROLLBACK command being used in response to SYNCPOINT on an APPC mapped conversation. The figure also shows the conversation state before each command and the state and EIB fields set after each command.

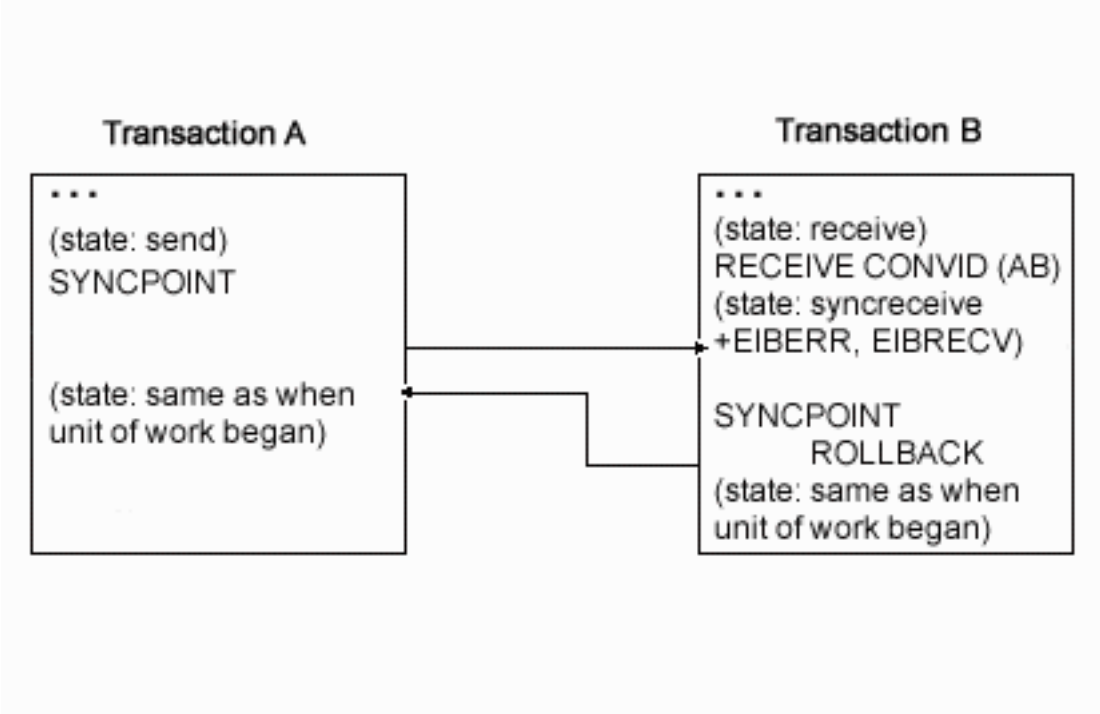


Figure 27. SYNCPOINT ROLLBACK in response to SYNCPOINT on an APPC mapped conversation

EXEC CICS SYNCPOINT ROLLBACK in response to EXEC CICS ISSUE PREPARE

Figure 28 on page 129 illustrates a SYNCPOINT ROLLBACK command being used in response to ISSUE PREPARE on an APPC mapped conversation. The figure also shows the conversation state before each command and the state and EIB fields set after each command.

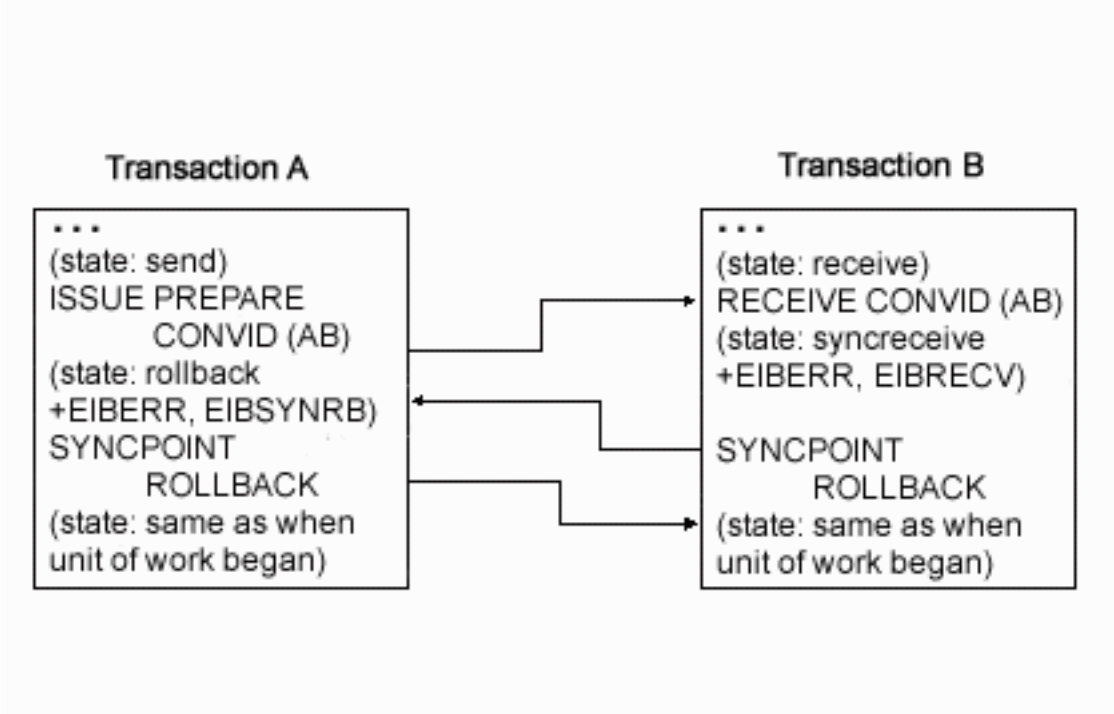


Figure 28. SYNCPOINT ROLLBACK in response to ISSUE PREPARE on an APPC mapped conversation

EXEC CICS ISSUE ERROR in response to EXEC CICS SYNCPOINT

Figure 29 on page 130 illustrates an ISSUE ERROR command being used in response to SYNCPOINT on an APPC mapped conversation. The figure also shows the conversation state before each command and the state and EIB fields set after each command. You can also send ISSUE ERROR before receiving SYNCPOINT; but this is not shown, because the results are the same.

It is pointless to use ISSUE ERROR as a response to SYNCPOINT, because this causes the syncpoint initiator to discard all data transmitted with the ISSUE ERROR by the syncpoint agent. To safeguard integrity, the syncpoint agent has to issue a SYNCPOINT ROLLBACK command.

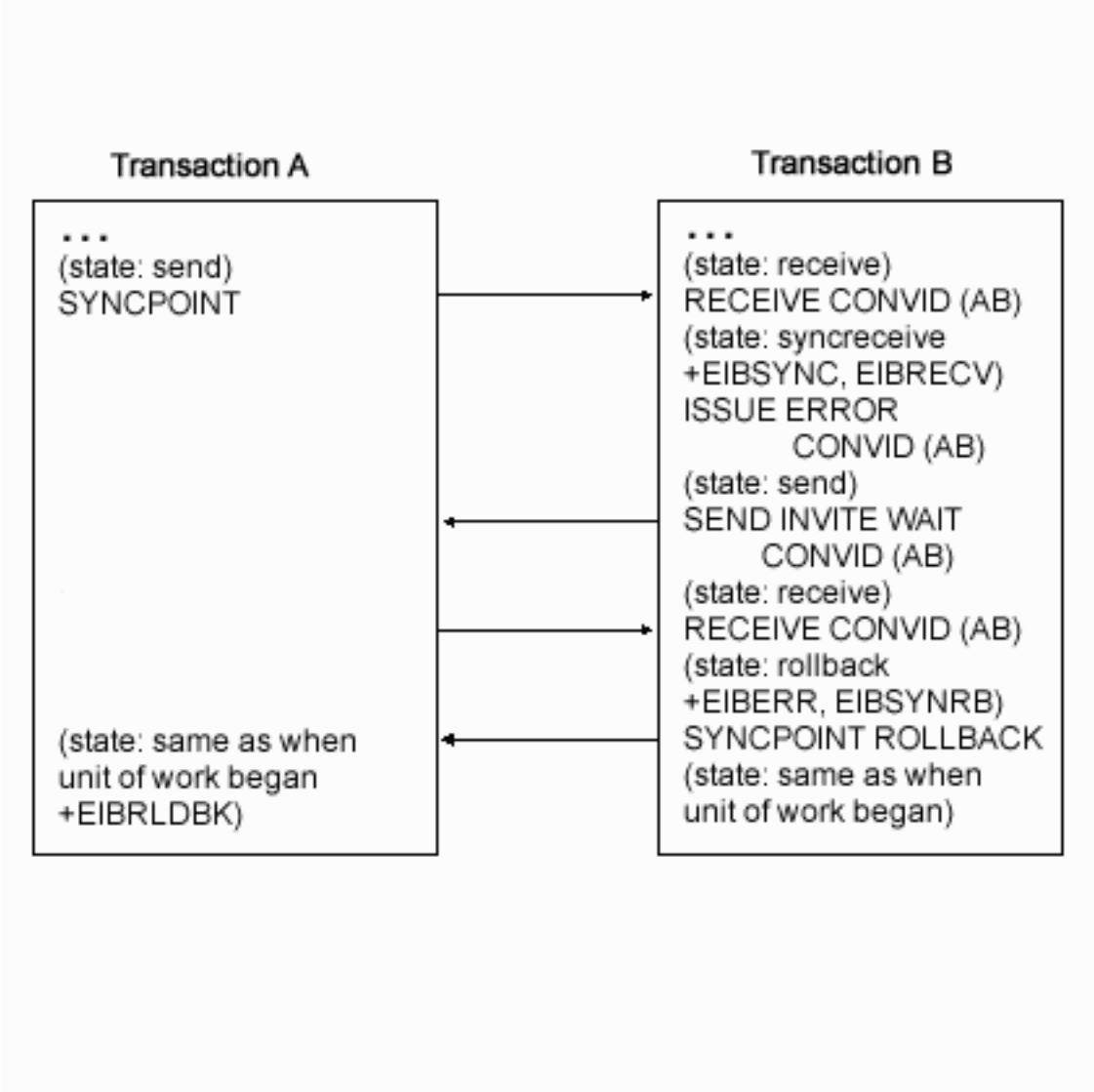


Figure 29. ISSUE ERROR in response to SYNCPOINT on an APPC mapped conversation

EXEC CICS ISSUE ERROR in response to EXEC CICS ISSUE PREPARE

Figure 30 on page 131 illustrates an ISSUE ERROR command being used in response to ISSUE PREPARE on an APPC mapped conversation. The figure also shows the conversation state before each command and the state and EIB fields set after each command. You can also send ISSUE ERROR before receiving ISSUE PREPARE; but this is not shown, because the results are the same.

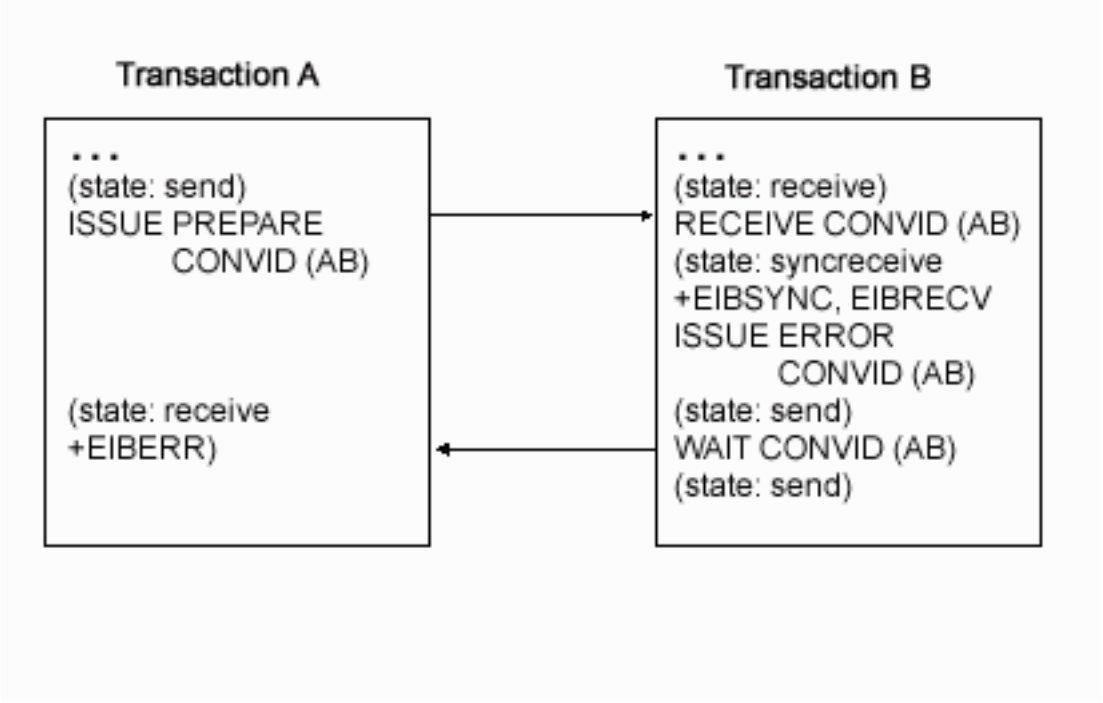


Figure 30. ISSUE ERROR in response to ISSUE PREPARE on an APPC mapped conversation

EXEC CICS ISSUE ABEND in response to EXEC CICS SYNCPOINT

Figure 31 on page 132 illustrates an ISSUE ABEND command being used in response to SYNCPOINT on an APPC mapped conversation. The figure also shows the conversation state before each command and the state and EIB fields set after each command. You can also send ISSUE ABEND before receiving SYNCPOINT; but this is not shown, because the results are the same.

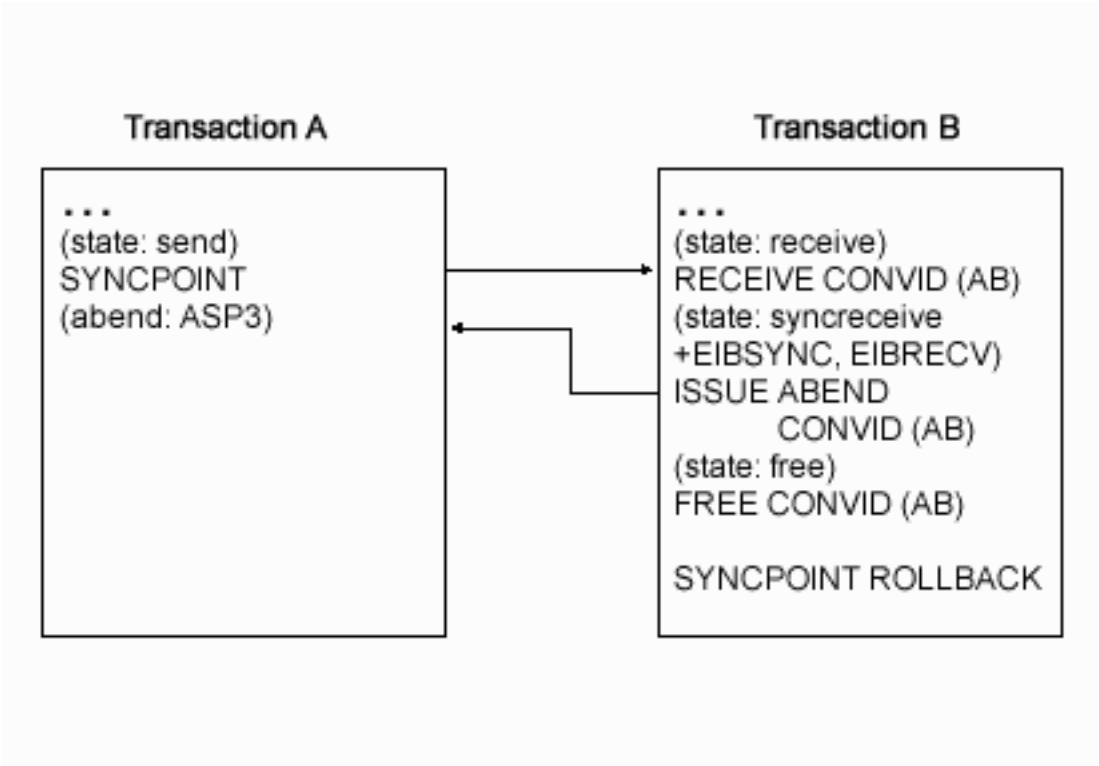


Figure 31. ISSUE ABEND in response to SYNCPOINT on an APPC mapped conversation

EXEC CICS ISSUE ABEND in response to EXEC CICS ISSUE PREPARE

Figure 32 on page 133 illustrates an ISSUE ABEND command being used in response to ISSUE PREPARE on an APPC mapped conversation. The figure also shows the conversation state before each command and the state and EIB fields set after each command. You can also send ISSUE ABEND before receiving ISSUE PREPARE; but this is not shown, because the results are the same.

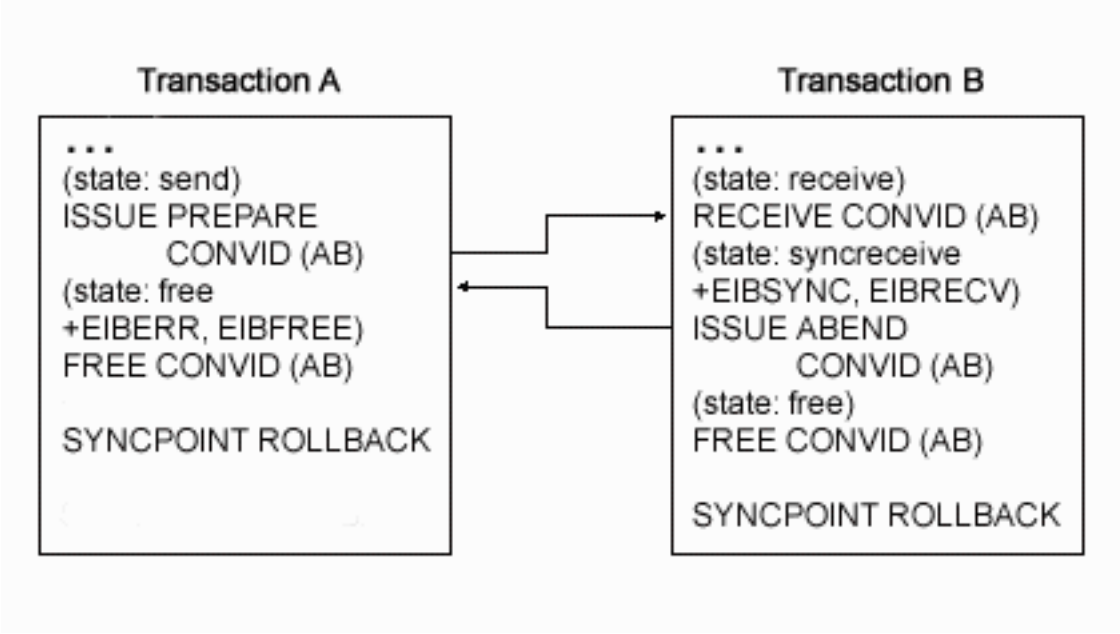


Figure 32. ISSUE ABEND in response to ISSUE PREPARE on an APPC mapped conversation

Session failure in response to EXEC CICS SYNCPOINT

Figure 33 on page 134 and Figure 34 illustrate what happens if the session fails before or after a SYNCPOINT command issued in response to SYNCPOINT on an APPC mapped conversation. The figures also show the conversation state before each command and the state and EIB fields set after each command.

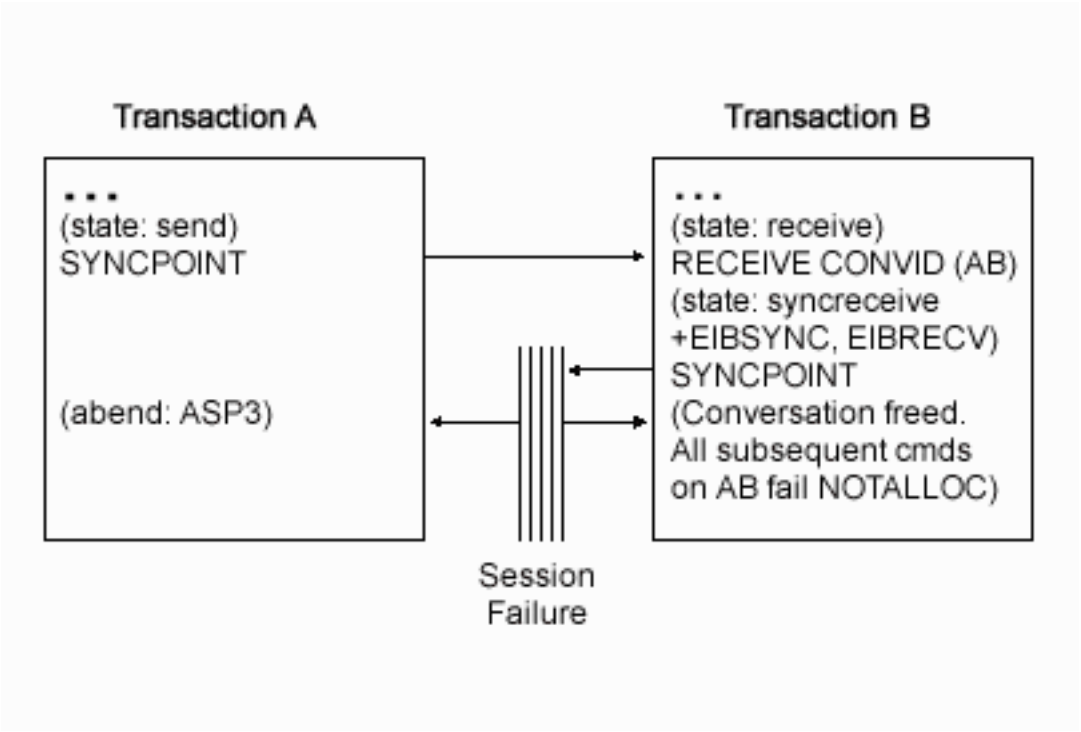


Figure 33. Session failure before SYNCPOINT in response to SYNCPOINT on an APPC mapped conversation

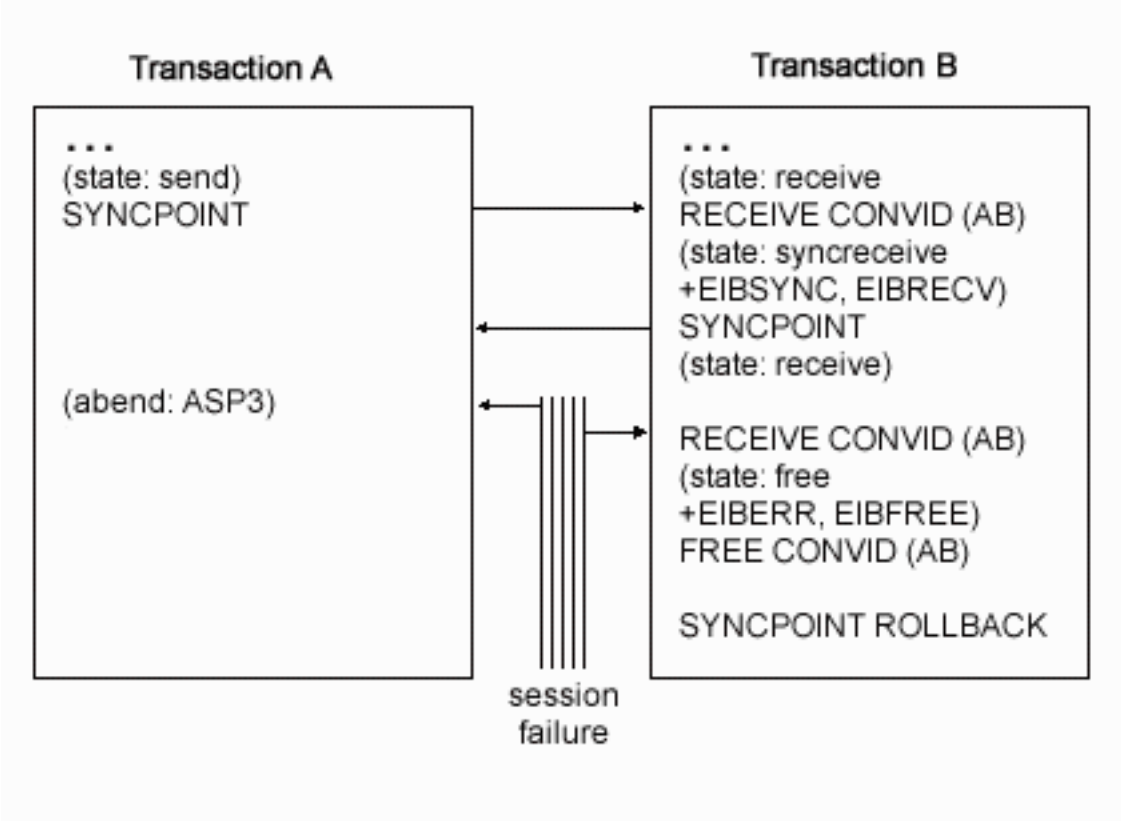


Figure 34. Session failure after SYNCPOINT in response to SYNCPOINT on an APPC mapped conversation

Session failure in response to EXEC CICS ISSUE PREPARE

Figure 35 illustrates what happens if the session fails after ISSUE PREPARE is received by transaction B and before the SYNCPOINT response is received by transaction A on an APPC mapped conversation. The figure also shows the conversation state before each command and the state and EIB fields set after each command.

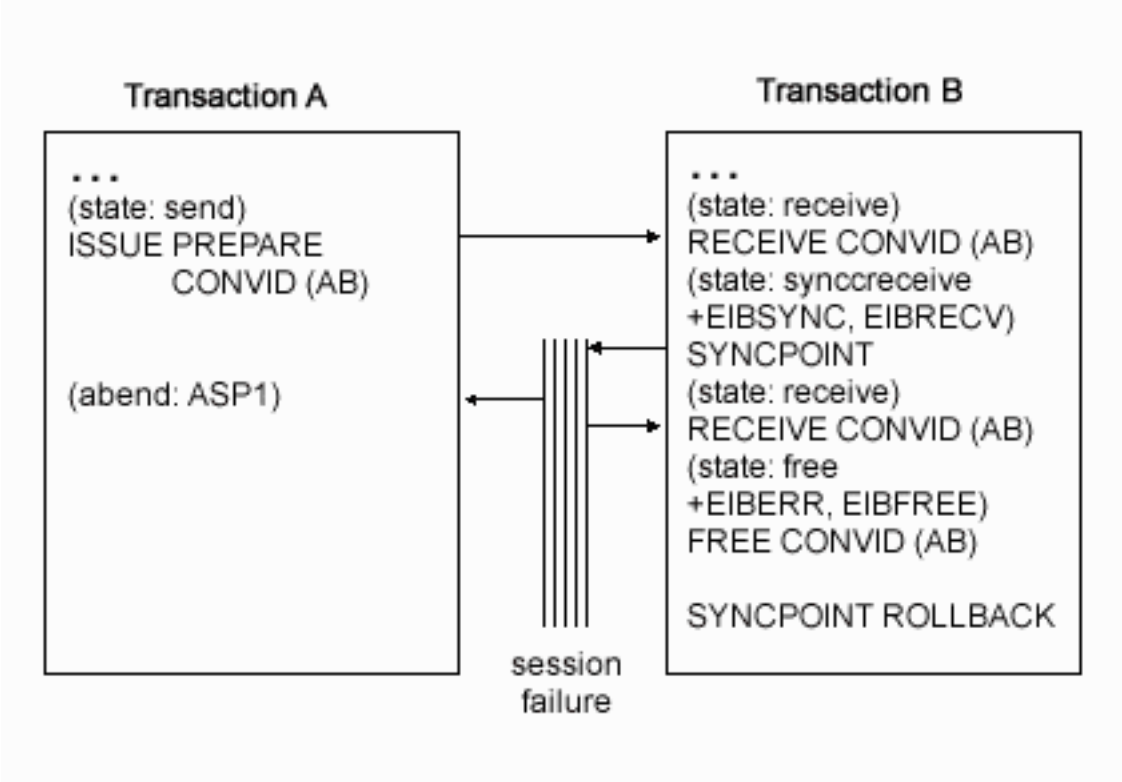


Figure 35. Session failure during SYNCPOINT in response to ISSUE PREPARE on an APPC mapped conversation

Session failure in response to EXEC CICS SYNCPOINT ROLLBACK

Figure 36 on page 136 illustrates what happens if the session fails after SYNCPOINT ROLLBACK is received and before the response is issued on an APPC mapped conversation. The figure also shows the conversation state before each command and the state and EIB fields set after each command.

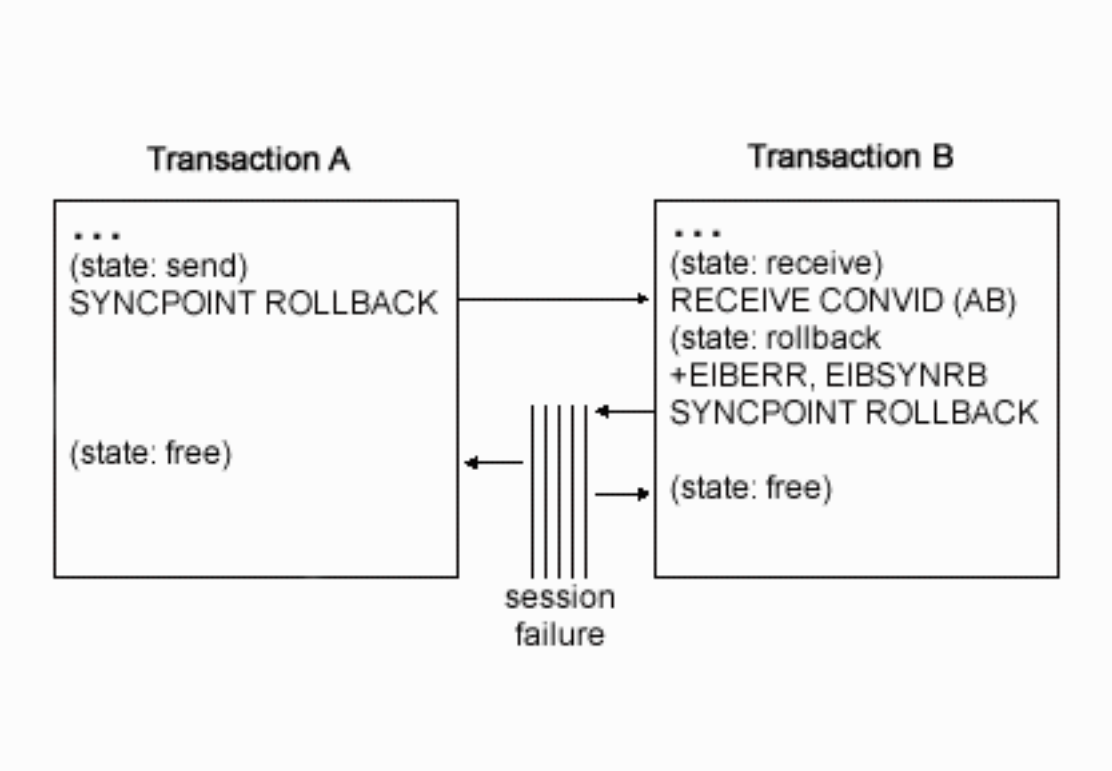


Figure 36. Session failure during SYNCPOINT ROLLBACK in response to SYNCPOINT ROLLBACK on an APPC mapped conversation

Synchronizing three or more CICS systems

This section gives examples of how to commit and back out recoverable resources affected by three or more DTP transactions connected on conversations at sync level 2.

EXEC CICS SYNCPOINT in response to EXEC CICS SYNCPOINT

Figure 37 on page 137 shows the sequence of events for a successful syncpoint involving six conversing transactions. It illustrates the states and actions that occur when transactions issue SYNCPOINT requests. To write successful distributed applications you do not need to understand all the data flows that take place during a distributed syncpoint. In this example, the programmer is concerned only with issuing SYNCPOINT in response to finding a conversation in **syncreceive state** (state 9).

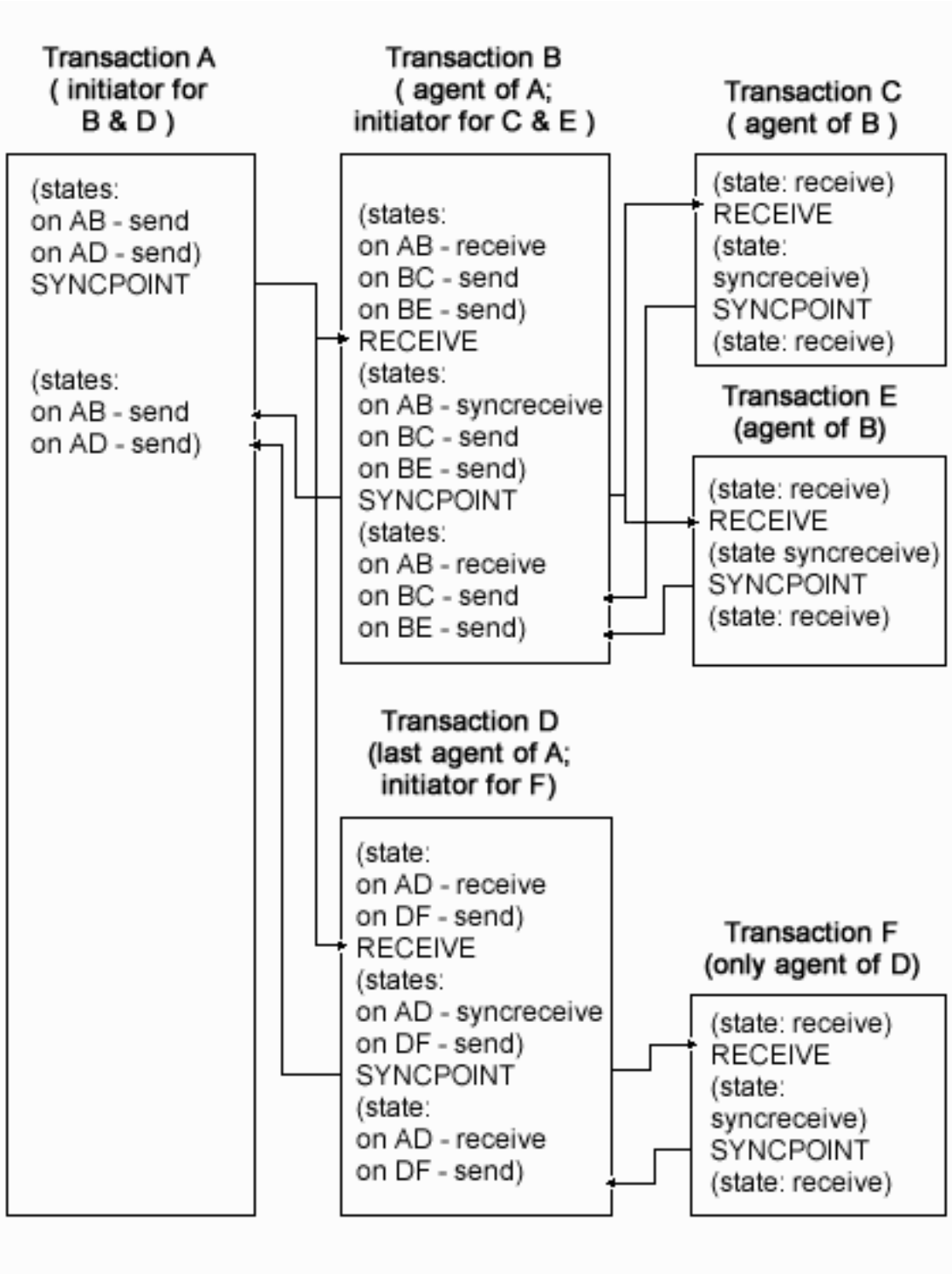


Figure 37. A distributed syncpoint with all partners running on CICS/400 Version 5

1. Transaction A, which is in **send state** (state 2) on its conversations with transactions B and D, decides to end the distributed unit of work, and therefore issues a SYNCPOINT command.
2. Transaction B sees that its half of its conversation with transaction A is in **syncreceive state** (state 9), so it issues a SYNCPOINT command. Transaction B is responding to a request from transaction A, but it also becomes the syncpoint

initiator for transactions C and E, and must ensure that its conversations with these transactions are in a valid state for issuing a SYNCPOINT command. In this example, they are both in **send state** (state 2).

3. Transaction C sees that its half of its conversation with transaction B is in **syncreceive state** (state 9), so it issues a SYNCPOINT command.
4. Transaction E sees that its half of its conversation with transaction B is in **syncreceive state** (state 9), so it issues a SYNCPOINT command.
5. Transaction D sees that its half of its conversation with transaction A is in **syncreceive state** (state 9), so it issues a SYNCPOINT command. Transaction D is responding to a request from transaction A, but it also becomes the syncpoint initiator for transaction F, and must ensure that its conversation with this transaction is in a valid state for issuing a SYNCPOINT command. In this example, it is in **send state** (state 2).
6. Transaction F sees that its half of its conversation with transaction D is in **syncreceive state** (state 9), so it issues a SYNCPOINT command.
7. All the transactions have now indicated, by issuing SYNCPOINT commands, that they are ready to commit their changes. This process begins with transaction F, which has no agents and has responded to "request commit" by issuing a SYNCPOINT command.
8. The distributed syncpoint is complete and control returns to transaction A following the SYNCPOINT command.

The previous discussion of the SYNCPOINT command assumed that all the agent transactions were ready to take a syncpoint by issuing SYNCPOINT when their conversation entered **syncreceive state** (state 9).

If, however, an agent has detected an error, it can reject the syncpoint request with one of the following commands:

- SYNCPOINT ROLLBACK (preferred response)
- ISSUE ERROR
- ISSUE ABEND

The SYNCPOINT ROLLBACK command enables a transaction to initiate a backout operation across the entire distributed unit of work. When it is issued in response to a syncpoint request, it has the following effects:

1. Any changes made to recoverable resources by the transaction that issues the rollback request are backed out.
2. The syncpoint initiator is also backed out (EIBRLDBK set).

This causes the syncpoint initiator to initiate a backout operation across the distributed unit of work.

EXEC CICS SYNCPOINT ROLLBACK in response to EXEC CICS SYNCPOINT

Figure 38 on page 139 shows the same distributed process as Figure 37 on page 137. Six transactions are engaged in related conversations. Transaction A (the first initiator) has two conversations: one with transaction B, and the other with transaction D. Transaction B has three conversations: one on its principal facility (with transaction A), another with transaction C, and another with transaction E. Transactions C and E each have one conversation: on their principal facility (with transaction B). Transaction D has two conversations: one on its principal facility

(with transaction A), and the other with transaction F. Transaction F has one conversation: on its principal facility (with transaction D).

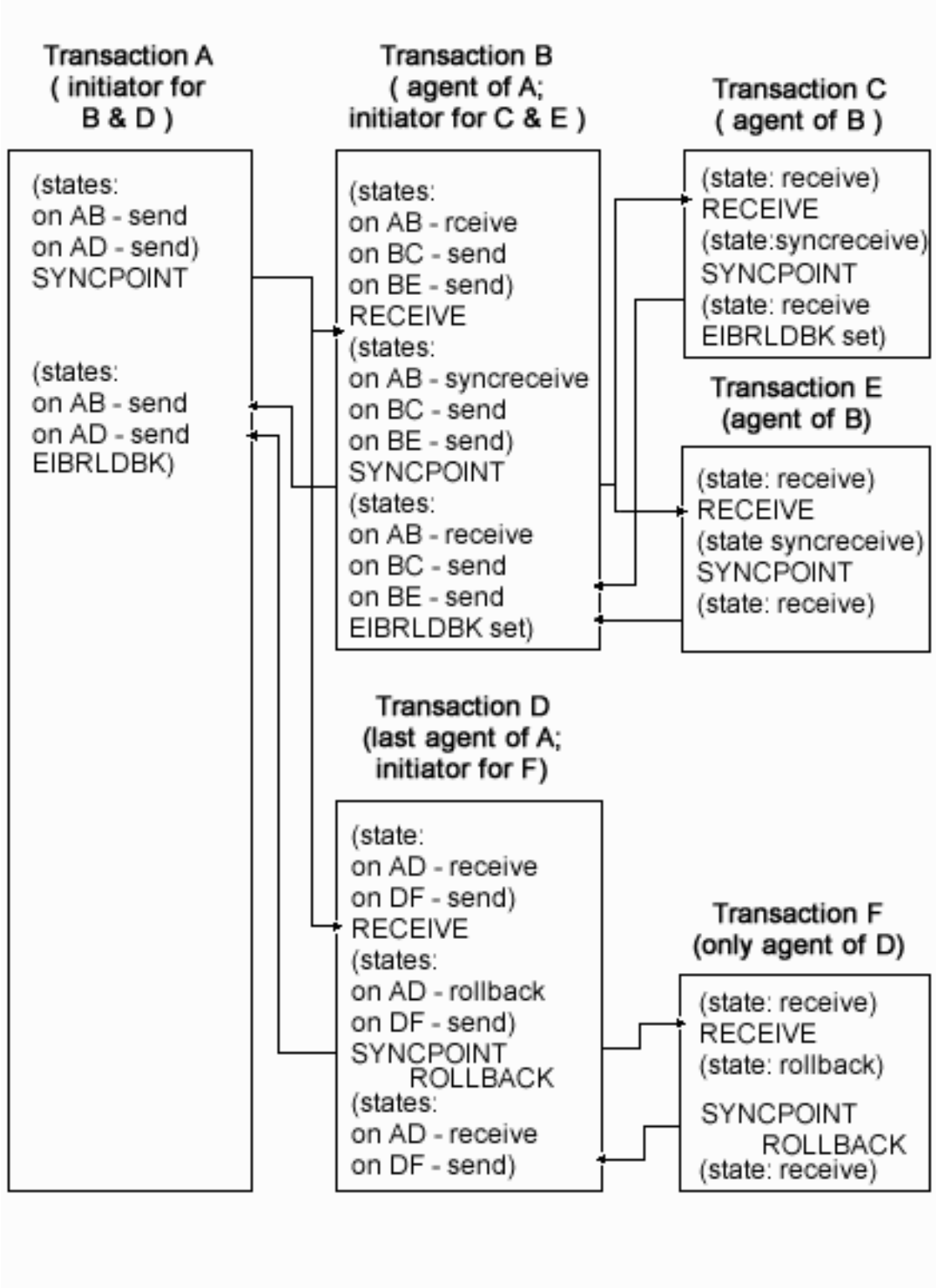


Figure 38. Rollback during distributed syncpointing

As in Figure 37 on page 137, transaction A (while in **send state**, state 2) issues the SYNCPOINT command, and CICS initiates a chain of events. Here, however,

transaction E has detected an error that makes it unable to commit, and it issues SYNCPOINT ROLLBACK when it detects that the conversation on its principal facility is in **syncreceive state** (state 9, EIBSYNC is also set). This causes any changes that transaction E has made to be backed out, and initiates a distributed rollback.

Transactions B, C, and A are rolled back (EIBRLDBK set). Transaction D senses that the conversation on its principal facility is in **rollback state** (state 13, EIBSYNRB is also set), and issues a SYNCPOINT ROLLBACK command. Transaction F too senses that the conversation on its principal facility is in **rollback state**, and issues a SYNCPOINT ROLLBACK command. The distributed rollback is now complete.

Session failure and the in-doubt period

During the period between the sending of the syncpoint request to the partner system and the receipt of the reply, the local system does not know whether the partner system has committed the change. This is known as the **in-doubt period**. If the intersystem session fails during this period, the local CICS system cannot tell whether the partner system has committed or backed out its resource changes.

This situation could occur for situations other than DTP and is discussed in the "Recovery and restart" section of the *CICS/ESA Intercommunication Guide*.

What really flows between APPC systems

This section describes the commit protocols that flow between APPC systems during a syncpoint. The arrows in the diagrams show the syncpoint flows in more detail than in the figures earlier in this chapter.

First, consider a simple distributed process involving only one conversation, as in Figure 39 on page 141. Here is what happens:

1. The syncpoint initiator sends a "commit" request to the syncpoint agent.
2. The syncpoint agent commits all changes it made to recoverable resources, and responds with "committed".
3. The syncpoint initiator then commits its changes, and the LUW is complete.

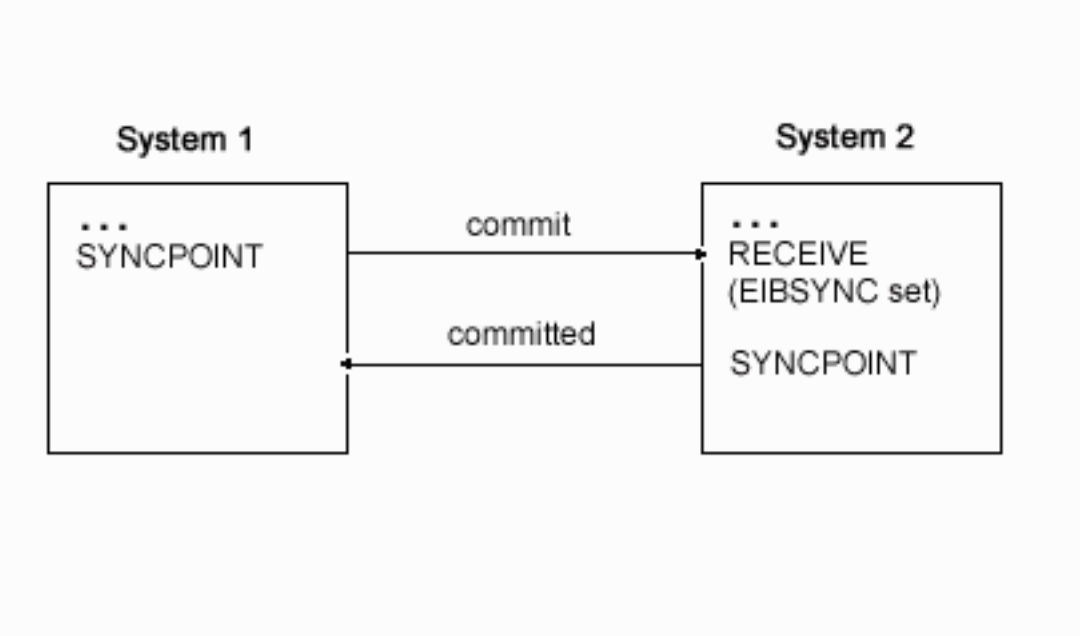


Figure 39. Syncpoint flows in a single conversation

When the syncpoint agent has a conversation with a third transaction, Figure 40 on page 142 shows the flows that occur. Here is what happens:

1. The syncpoint initiator sends a “commit” request to its agent.
2. The agent becomes the initiator on the conversation to its agent, and sends a “commit” request.
3. The second agent commits first and responds with “committed”.
4. The first agent commits and sends “committed” to the initiator.
5. The initiator commits.

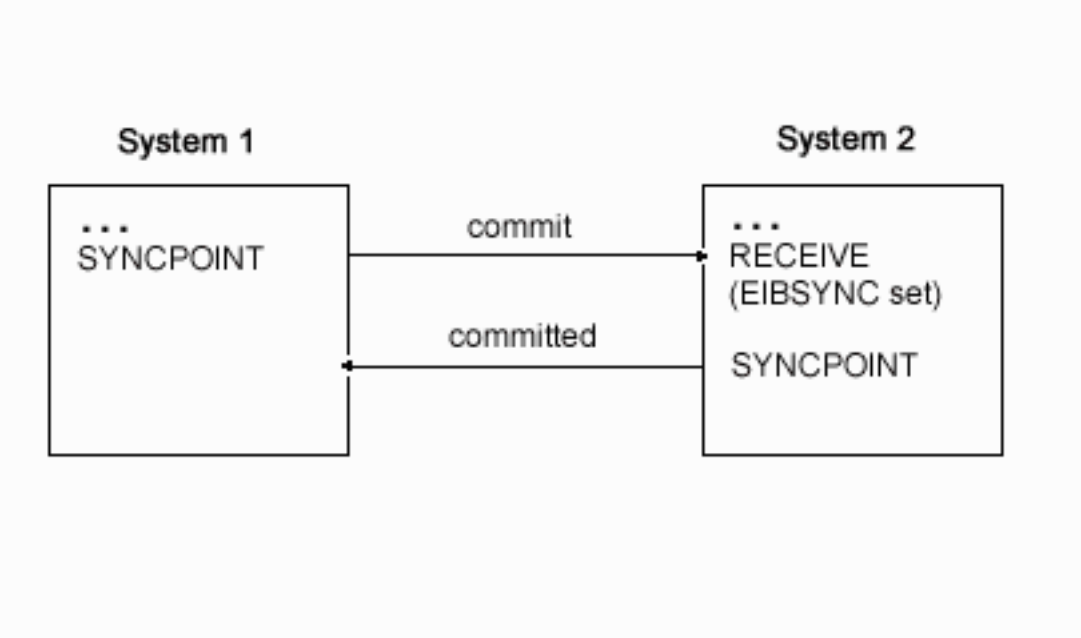


Figure 40. Syncpoint flows in concurrent conversations

When the syncpoint initiator has two concurrent conversations, the flows involved are shown in Figure 41 on page 143. Here is what happens:

1. The syncpoint initiator sends a “prepare” request to all its agents except one.
2. The agent receiving “prepare” responds by sending a “commit” request to the initiator.
3. When all the “prepare” requests have been sent, and the “commit” requests received, the initiator sends a “commit” request to its last agent.
4. The initiator receives “committed” from the last agent.
5. The initiator sends “committed” to the remaining agents.
6. The agents respond “forget” to indicate that they do not need to be resynchronized.

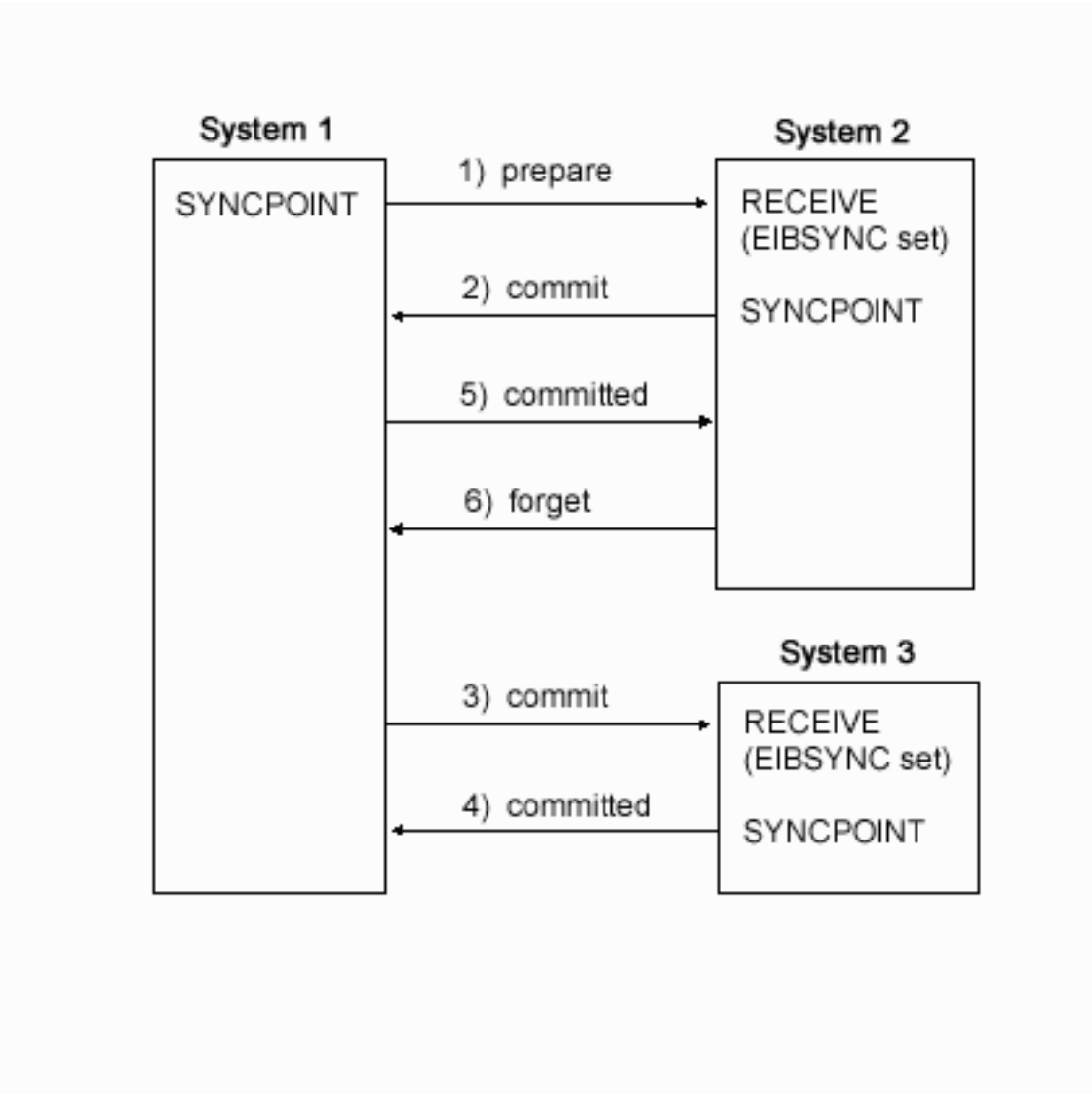


Figure 41. Syncpoint flows in concurrent conversations with one initiator. The initiator uses only SYNCPOINT.

If the syncpoint initiator decides to prepare the conversation with system 2 explicitly before issuing a syncpoint, the flows involved are shown in Figure 42 on page 144. You will notice that, although the application program in system 1 issues extra commands, the flows across the links are exactly the same as those in the previous example. Using the ISSUE PREPARE command gives the application the opportunity to "change its mind" and rollback, depending on the response to ISSUE PREPARE.

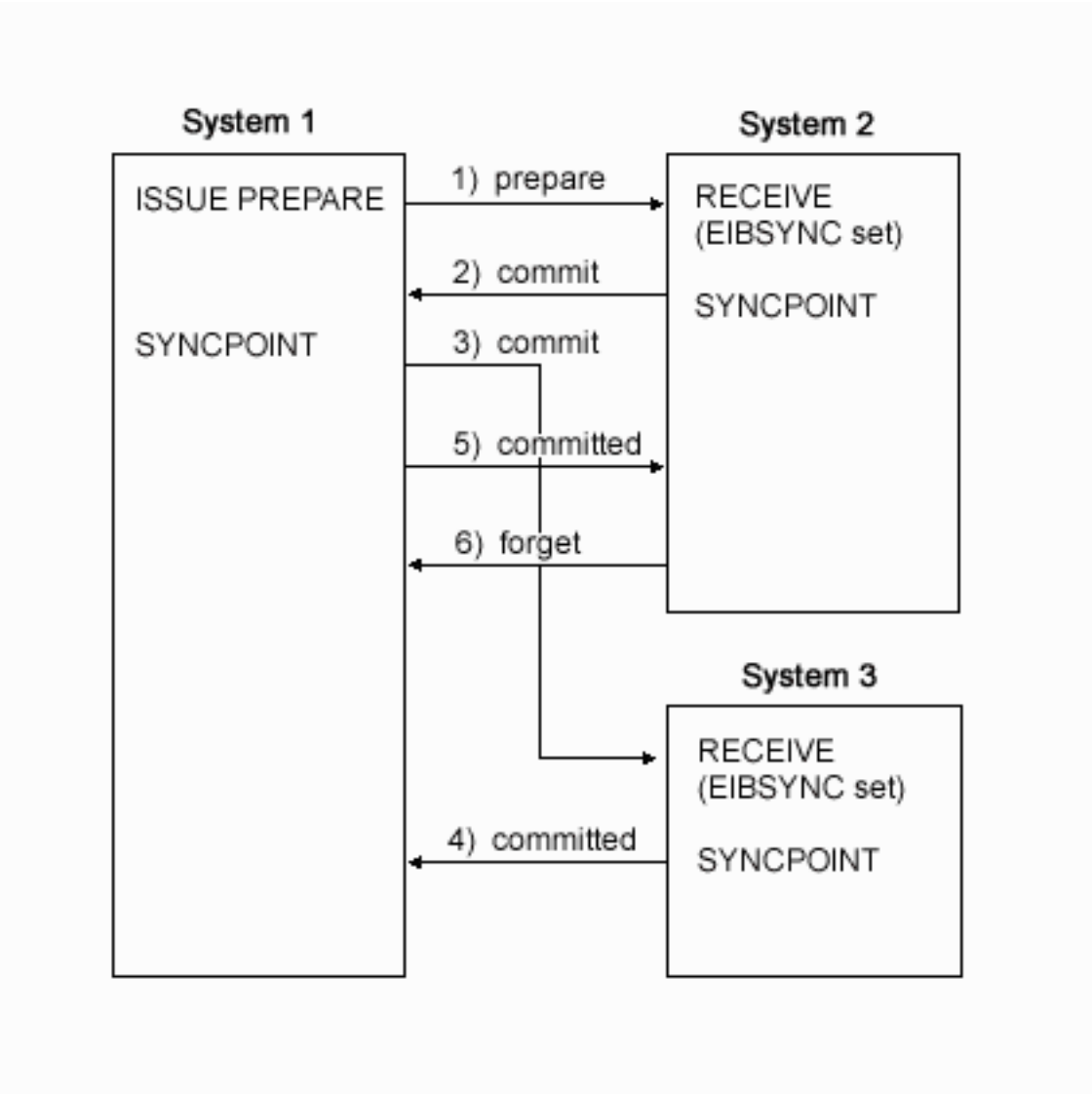


Figure 42. Syncpoint flows in concurrent conversations with one initiator. The initiator uses ISSUE PREPARE before SYNCPOINT.

For further information on the flows in a distributed process, see the *SNA LU6.2 Reference: Peer Protocols* book.

Chapter 13. State transitions in APPC mapped conversations

General-use programming interface

This chapter shows the state transitions that occur when transactions engage in APPC mapped conversations under the EXEC CICS API. The state transitions are presented in the form of state tables; and there is one table for each of the three allowable sync levels. The state tables show which commands a transaction can issue while the conversation is in any given state. They also show how the conversation state changes as a result of any command.

The guidance information in this chapter is presented in the following sequence:

- "The state tables for APPC mapped conversations"
- "Testing the conversation state" on page 154

The state tables for APPC mapped conversations

The state tables provide the following information for writing a DTP program. Firstly, they show which commands can be issued from each conversation state. Secondly, they show the state transitions that can occur and the EIB fields that can be set as a result of issuing a command.

How to use the state tables

The commands you can issue, coupled with the EIB flags that can be set after execution, are shown in column 1 down the left side of each table. Alongside each command, in column 2, the EIB fields shown are in the order in which the application should test them. The possible conversation states are shown across the top of the table. The states correspond to the columns of the table. The intersection of row (command and EIB flag) and column (state) represents the state transition, if any, that occurs when that command returning a particular EIB flag is issued in that state.

A number at an intersection indicates the state number of the next state. Other symbols represent other conditions, as follows:

Symbol	Meaning
N/A	Cannot occur.
x	The EIB flag is any one that has not been covered in earlier rows, or it is irrelevant (but see the note on EIBSIG if you want to use ISSUE SIGNAL).
Ab	The command is not valid in this state. Issuing a command in a state in which it is not valid usually causes an ATCV abend.
=	Remains in current state.
End	End of conversation.

APPC mapped conversations at sync level 0

Command issued	EIB flag returned[19]	ALLOCATED[26] State 1	SEND State 2	PEND-RECEIVE State 3	PEND-FREE State 4	RECEIVE State 5	CONF-RECEIVE State 6
CONNECT PROCESS	EIBERR + EIBFREE	12	Ab	Ab	Ab	Ab	N/A
CONNECT PROCESS[28]	x	2	Ab	Ab	Ab	Ab	N/A
EXTRACT PROCESS[20]	x	=	=	=	=	=	N/A
EXTRACT ATTRIBUTES	x	=	=	=	=	=	N/A
SEND (any valid form)	EIBERR + EIBFREE	Ab	12	Ab	Ab	Ab	N/A
SEND (any valid form)	EIBERR	Ab	5	Ab	Ab	Ab	N/A
SEND INVITE WAIT	x	Ab	5	Ab	Ab	Ab	N/A
SEND INVITE	x	Ab	3	Ab	Ab	Ab	N/A
SEND LAST WAIT	x	Ab	12	Ab	Ab	Ab	N/A
SEND LAST	x	Ab	4	Ab	Ab	Ab	N/A
SEND WAIT	x	Ab	=	Ab	Ab	Ab	N/A
SEND	x	Ab	=	Ab	Ab	Ab	N/A
RECEIVE	EIBERR + EIBFREE	Ab	12[22]	12[25]	Ab	12	N/A
RECEIVE	EIBERR	Ab	5[22]	5[25]	Ab	=	N/A
RECEIVE	EIBREE	Ab	Ab	12	N/A		
RECEIVE	EIBRECV	Ab	5[22]	5[25]	Ab	=	N/A
RECEIVE NOTRUNCATE[21]	EIBCOMPL[21]	Ab	5[22]	5[25]	Ab	=	N/A
RECEIVE	x	Ab	=[22]	2[25]	Ab	2	N/A
CONVERSE[23]	EIB flags and states as for RECEIVE						
ISSUE ERROR	EIBFREE	Ab	12	12	Ab	12	N/A
ISSUE ERROR	x	Ab	=	2	Ab	2	N/A
ISSUE ERROR	x	Ab	12	12	12	12	N/A
ISSUE ABEND	x	Ab	=	=	Ab	=	N/A
ISSUE SIGNAL[12]							
WAIT CONVID	x	Ab	=	5	12	Ab	N/A
FREE	x	End	End[24]	Ab	End	Ab	N/A

Note: See 152 for footnotes.

CONF-SEND State 7	CONF-FREE State 8	SYNC-RECEIVE State 9	SYNC-SEND State 10	SYNC-FREE State 11	FREE State 12	ROLL-BACK State 13	Command returns
N/A	N/A	N/A	N/A	N/A	Ab	N/A	Immediately
N/A	N/A	N/A	N/A	N/A	Ab	N/A	Immediately
N/A	N/A	N/A	N/A	N/A	=	N/A	Immediately
N/A	N/A	N/A	N/A	N/A	=	N/A	Immediately
N/A	N/A	N/A	N/A	N/A	Ab	N/A	After error detected
N/A	N/A	N/A	N/A	N/A	Ab	N/A	After error detected
N/A	N/A	N/A	N/A	N/A	Ab	N/A	After data flows
N/A	N/A	N/A	N/A	N/A	Ab	N/A	After data buffered
N/A	N/A	N/A	N/A	N/A	Ab	N/A	After data flows
N/A	N/A	N/A	N/A	N/A	Ab	N/A	After data buffered
N/A	N/A	N/A	N/A	N/A	Ab	N/A	After data flows
N/A	N/A	N/A	N/A	N/A	Ab	N/A	After data buffered
N/A	N/A	N/A	N/A	N/A	Ab	N/A	After error detected
N/A	N/A	N/A	N/A	N/A	Ab	N/A	After error detected
N/A	N/A	N/A	N/A	N/A	Ab	N/A	After error detected
N/A	N/A	N/A	N/A	N/A	Ab	N/A	When data available
N/A	N/A	N/A	N/A	N/A	Ab	N/A	When data available
N/A	N/A	N/A	N/A	N/A	Ab	N/A	When data available
States as for RECEIVE							When data available
N/A	N/A	N/A	N/A	N/A	Ab	N/A	After response from partner
N/A	N/A	N/A	N/A	N/A	Ab	N/A	After response from partner
N/A	N/A	N/A	N/A	N/A	Ab	N/A	Immediately
N/A	N/A	N/A	N/A	N/A	Ab	N/A	Immediately
N/A	N/A	N/A	N/A	N/A	Ab	N/A	Immediately
N/A	N/A	N/A	N/A	N/A	End	N/A	Immediately

APPC mapped conversations at sync level 1

Command issued	EIB flag returned[19]	ALLOCATED[26] State 1	SEND State 2	PEND-RECEIVE State 3	PEND-FREE State 4	RECEIVE State 5	CONF-RECEIVE State 6
CONNECT PROCESS	EIBERR + EIBFREE	12	Ab	Ab	Ab	Ab	Ab
CONNECT PROCESS[28]	x	2	Ab	Ab	Ab	Ab	Ab
EXTRACT PROCESS[20]	x	Ab	=	=	=	=	=
EXTRACT ATTRIBUTES	x	=	=	=	=	=	=
SEND (any valid form)	EIBERR + EIBFREE	Ab	12	12	12	Ab	Ab
SEND (any valid form)	EIBERR	Ab	5	5	5	Ab	Ab
SEND INVITE WAIT	x	Ab	5	Ab	Ab	Ab	Ab
SEND INVITE CONFIRM	x	Ab	5	Ab	Ab	Ab	Ab
SEND INVITE	x	Ab	3	Ab	Ab	Ab	Ab
SEND LAST WAIT	x	Ab	12	Ab	Ab	Ab	Ab
SEND LAST CONFIRM	x	Ab	12	Ab	Ab	Ab	Ab
SEND LAST	x	Ab	4	Ab	Ab	Ab	Ab
SEND WAIT	x	Ab	=	Ab	Ab	Ab	Ab
SEND CONFIRM	x	Ab	=	5	12[29]	Ab	Ab
SEND	x	Ab	=	Ab	Ab	Ab	Ab
RECEIVE	EIBERR + EIBFREE	Ab	12[22]	12[25]	Ab	12	Ab
RECEIVE	EIBERR	Ab	5[22]	5[25]	Ab	=	Ab
RECEIVE	EIBCONF + EIBREE	Ab	8[22]	8[25]	Ab	8	Ab
RECEIVE	EIBCONF + EIBRECV	Ab	6[22]	6[25]	Ab	6	Ab
RECEIVE	EIBCONF	Ab	7[22]	7[25]	Ab	7	Ab
RECEIVE	EIBFREE	Ab	12[22]	12[25]	Ab	12	Ab
RECEIVE	EIBRECV	Ab	5[22]	5[25]	Ab	=	Ab
RECEIVE NOTRUNCATE[21]	EIBCOMPL[21]	Ab	5[22]	5[25]	Ab	=	Ab
RECEIVE	x	Ab	=[22]	2[25]	Ab	2	Ab
CONVERSE[23]	EIB flags and states as for RECEIVE						
ISSUE CONFIRMATION	x	Ab	Ab	Ab	Ab	Ab	5
ISSUE ERROR	EIBFREE	Ab	12	12	Ab	12	12
ISSUE ERROR	x	Ab	=	2	Ab	2	2
ISSUE ABEND	x	Ab	12	12	12	12	12
ISSUE SIGNAL[27]	x	Ab	=	=	Ab	=	=
WAIT CONVID	x	Ab	=	5	12	Ab	Ab
FREE	x	End	End[24]	Ab	End	Ab	Ab

Note: See 152 for footnotes.

CONF-SEND State 7	CONF-FREE State 8	SYNC-RECEIVE State 9	SYNC-SEND State 10	SYNC-FREE State 11	FREE State 12	ROLL-BACK State 13	Command returns
Ab	Ab	N/A	N/A	N/A	Ab	N/A	Immediately
Ab	Ab	N/A	N/A	N/A	Ab	N/A	Immediately
=	=	N/A	N/A	N/A	=	N/A	Immediately
=	=	N/A	N/A	N/A	=	N/A	Immediately
Ab	Ab	N/A	N/A	N/A	Ab	N/A	After error flow detected
Ab	Ab	N/A	N/A	N/A	Ab	N/A	After error flow detected
Ab	Ab	N/A	N/A	N/A	Ab	N/A	After data flows
Ab	Ab	N/A	N/A	N/A	Ab	N/A	After response from partner
Ab	Ab	N/A	N/A	N/A	Ab	N/A	After data buffered
Ab	Ab	N/A	N/A	N/A	Ab	N/A	After data flows
Ab	Ab	N/A	N/A	N/A	Ab	N/A	After response from partner
Ab	Ab	N/A	N/A	N/A	Ab	N/A	After data buffered
Ab	Ab	N/A	N/A	N/A	Ab	N/A	After data flows
Ab	Ab	N/A	N/A	N/A	Ab	N/A	After response from partner
Ab	Ab	N/A	N/A	N/A	Ab	N/A	After data buffered
Ab	Ab	N/A	N/A	N/A	Ab	N/A	After error detected
Ab	Ab	N/A	N/A	N/A	Ab	N/A	After error detected
Ab	Ab	N/A	N/A	N/A	Ab	N/A	After confirm flow detected
Ab	Ab	N/A	N/A	N/A	Ab	N/A	After confirm flow detected
Ab	Ab	N/A	N/A	N/A	Ab	N/A	After confirm flow detected
Ab	Ab	N/A	N/A	N/A	Ab	N/A	After error detected
Ab	Ab	N/A	N/A	N/A	Ab	N/A	When data available
Ab	Ab	N/A	N/A	N/A	Ab	N/A	When data available
Ab	Ab	N/A	N/A	N/A	Ab	N/A	When data available
States as for RECEIVE							When data available
2	12	N/A	N/A	N/A	Ab	N/A	Immediately
12	12	N/A	N/A	N/A	Ab	N/A	After response from partner
2	2	N/A	N/A	N/A	Ab	N/A	After response from partner
12	12	N/A	N/A	N/A	Ab	N/A	Immediately
=	=	N/A	N/A	N/A	Ab	N/A	Immediately
Ab	Ab	N/A	N/A	N/A	Ab	N/A	Immediately
Ab	Ab	N/A	N/A	N/A	End	N/A	Immediately

table continued ...

APPC mapped conversations at sync level 2

Command issued	EIB flag returned[19]	ALLOCATED[26] State 1	SEND State 2	PEND-RECEIVE State 3	PEND-FREE State 4	RECEIVE State 5	CONF-RECEIVE State 6
CONNECT PROCESS	EIBERR + EIBFREE	12	Ab	Ab	Ab	Ab	Ab
CONNECT PROCESS[28]	x	2	Ab	Ab	Ab	Ab	Ab
EXTRACT PROCESS[20]	x	=	=	=	=	=	=
EXTRACT ATTRIBUTES	x	=	=	=	=	=	=
SEND (any valid form)	EIBERR + EIBSYNRB	Ab	13	13	13	Ab	Ab
SEND (any valid form)	EIBERR + EIBFREE	Ab	12	12	12	Ab	Ab
SEND (any valid form)	EIBERR	Ab	5	5	5	Ab	Ab
SEND INVITE WAIT	x	Ab	5	Ab	Ab	Ab	Ab
SEND INVITE CONFIRM	x	Ab	5	Ab	Ab	Ab	Ab
SEND INVITE	x	Ab	3	Ab	Ab	Ab	Ab
SEND LAST WAIT[30]	x	Ab	Ab	Ab	Ab	Ab	Ab
SEND LAST CONFIRM[30]	x	Ab	Ab	Ab	Ab	Ab	Ab
SEND LAST	x	Ab	4	Ab	Ab	Ab	Ab
SEND WAIT	x	Ab	=	Ab	Ab	Ab	Ab
SEND CONFIRM	x	Ab	=	5[29]	12[29]	Ab	Ab
SEND	x	Ab	=	Ab	Ab	Ab	Ab
RECEIVE	EIBERR + EIBSYNRB	Ab	13[22]	13[25]	Ab	13	Ab
RECEIVE	EIBERR + EIBFREE	Ab	12[22]	12[25]	Ab	12	Ab
RECEIVE	EIBERR	Ab	5[22]	5[25]	Ab	=	Ab
RECEIVE	EIBSYNC + EIBFREE	Ab	11[22]	11[25]	Ab	11	Ab
RECEIVE	EIBSYNC + EIBRECV	Ab	9[22]	9[25]	Ab	9	Ab
RECEIVE	EIBSYNC	Ab	10[22]	10[25]	Ab	10	Ab
RECEIVE	EIBCONF + EIBREE	Ab	8[22]	8[25]	Ab	8	Ab
RECEIVE	EIBCONF + EIBRECV	Ab	6[22]	6[25]	Ab	6	Ab
RECEIVE	EIBCONF	Ab	7[22]	7[25]	Ab	7	Ab
RECEIVE	EIBFREE	Ab	12[22]	12[25]	Ab	12	Ab
RECEIVE	EIBRECV	Ab	5[22]	5[25]	Ab	=	Ab
RECEIVE NOTRUNCATE[21]	EIBCOMPL[21]	Ab	5[22]	5[25]	Ab	=	Ab
RECEIVE	x	Ab	=[22]	2[25]	Ab	2	Ab
CONVERSE[23]	EIB flags and states as for RECEIVE						

Note: See 152 for footnotes.

CONF-SEND State 7	CONF-FREE State 8	SYNC-RECEIVE State 9	SYNC-SEND State 10	SYNC-FREE State 11	FREE State 12	ROLL-BACK State 13	Command returns
Ab	Ab	Ab	Ab	Ab	Ab	Ab	Immediately
Ab	Ab	Ab	Ab	Ab	Ab	Ab	Immediately
=	=	=	=	=	=	=	Immediately
=	=	=	=	=	=	=	Immediately
Ab	Ab	Ab	Ab	Ab	Ab	Ab	After error flow detected
Ab	Ab	Ab	Ab	Ab	Ab	Ab	After error flow detected
Ab	Ab	Ab	Ab	Ab	Ab	Ab	After error flow detected
Ab	Ab	Ab	Ab	Ab	Ab	Ab	After data flows
Ab	Ab	Ab	Ab	Ab	Ab	Ab	After response from partner
Ab	Ab	Ab	Ab	Ab	Ab	Ab	After data buffered
Ab	Ab	Ab	Ab	Ab	Ab	Ab	After data flows
Ab	Ab	Ab	Ab	Ab	Ab	Ab	After response from partner
Ab	Ab	Ab	Ab	Ab	Ab	Ab	After data buffered
Ab	Ab	Ab	Ab	Ab	Ab	Ab	After data flows
Ab	Ab	Ab	Ab	Ab	Ab	Ab	After response from partner
Ab	Ab	Ab	Ab	Ab	Ab	Ab	After data buffered
Ab	Ab	Ab	Ab	Ab	Ab	Ab	After rollback flow detected
Ab	Ab	Ab	Ab	Ab	Ab	Ab	After error detected
Ab	Ab	Ab	Ab	Ab	Ab	Ab	After error detected
Ab	Ab	Ab	Ab	Ab	Ab	Ab	After sync flow detected
Ab	Ab	Ab	Ab	Ab	Ab	Ab	After sync flow detected
Ab	Ab	Ab	Ab	Ab	Ab	Ab	After sync flow detected
Ab	Ab	Ab	Ab	Ab	Ab	Ab	After confirm flow detected
Ab	Ab	Ab	Ab	Ab	Ab	Ab	After confirm flow detected
							After confirm flow detected
Ab	Ab	Ab	Ab	Ab	Ab	Ab	After error flow detected
Ab	Ab	Ab	Ab	Ab	Ab	Ab	
Ab	Ab	Ab	Ab	Ab	Ab	Ab	When data available
Ab	Ab	Ab	Ab	Ab	Ab	Ab	When data available
Ab	Ab	Ab	Ab	Ab	Ab	Ab	When data available
States as for RECEIVE							When data available

table continued ...

APPC mapped conversations at sync level 2 (continued)

Command issued	EIB flag returned[19]	ALLOCATED[26] State 1	SEND State 2	PEND-RECEIVE State 3	PEND-FREE State 4	RECEIVE State 5	CONF-RECEIVE State 6
ISSUE CONFIRMATION	x	Ab	Ab	Ab	Ab	Ab	5
ISSUE ERROR	EIBFREE	Ab	12	12	Ab	12	12
ISSUE ERROR	x	Ab	=	2	Ab	2	2
ISSUE ABEND	x	Ab	12	12	12	12	12
ISSUE SIGNAL[27]	x	Ab	=	=	Ab	=	=
ISSUE PREPARE	EIBERR + EIBSYNRB	Ab[34]	13	13	13	Ab[34]	Ab[34]
ISSUE PREPARE	EIBERR + EIBFREE	Ab[34]	12	12	12	Ab[34]	Ab[34]
ISSUE PREPARE	EIBERR	Ab[34]	5	5	5	Ab[34]	Ab[34]
ISSUE PREPARE	x	Ab[34]	10[36]	9[36]	11[36]	Ab[34]	Ab[34]
SYNCPOINT[32]	EIBRLDBK	=	2 or 5[33]	2 or 5[33]	2 or 5[33]	Ab[35]	Ab[35]
SYNCPOINT[32]	x	=	=	5	12	Ab[35]	Ab[35]
SYNCPOINT ROLLBACK[32]	x	=	2 or 5[33]	2 or 5[33]	2 or 5[33]	2 or 5[33]	2 or 5[33]
WAIT CONVID	x	Ab	=	5	12	Ab	Ab
FREE	x	End	End[30]	Ab	End	Ab	Ab

Note:

19 EIBSIG has been omitted. This is because its use is optional and is entirely a matter of agreement between the two conversation partners. In the worst case, it can ocur at any time after every command that affects the EIB flags. However, used for the purpose for which it was intended, it usually occurs after a SEND command. Its priority in the order of testing depends on the role you give it in the application.

20 You can issue the EXTRACT PROCESS command from the back-end transaction only.

21 RECEIVE NOTRUNCATE returns a zero value in EIBCOMPL to indicate that the user buffer was too small to contain all the data received from the partner transaction. Normally, you would continue to issue RECEIVE NOTRUNCATE commands until the last section of data is passed to you, which is indicated by EIBCOMPL = X'FF'. If NOTRUNCATE is not specified, and the data area specified by the RECEIVE command is too smal to contain all the data received, CICS truncates the data and the sets the LENGERR condition.

22 Equivalend to SEND INVITE WAIT followed by RECEIVE.

23 Equivalent to SEND INVITE WAIT [FROM] followed by RECEIVE.

24 Equivalent to SEND LAST WAIT followed by FREE.

25 Equivalent to WAIT followed by RECEIVE.

26 Before a session is allocated, there is no conversation, and therefore no conversation state. The EXEC CICS ALLOCATE command does not appear in the tables. This is because each ALLOCATE gets a session to start a new conversation and does not affect any conversation that is already in progress. After ALLOCATE is successful, the front-end transaction starts the new conversation in **allocated state**.

27 ISSUE SIGNAL sets the partner's EIBSIG flag.

CONF-SEND State 7	CONF-FREE State 8	SYNC-RECEIVE State 9	SYNC-SEND State 10	SYNC-FREE State 11	FREE State 12	ROLL-BACK State 13	Command returns
2	12	Ab	Ab	Ab	Ab	Ab	Immediately
12	12	12	12	12	Ab	Ab	After response from partner
2	2	2	2	2	Ab	Ab	After response from partner
12	12	12	12	12	Ab	Ab	Immediately
=	=	=[31]	=[31]	=[31]	Ab	Ab	Immediately
Ab	Ab	Ab	Ab	Ab	Ab	Ab	After response from partner
Ab	Ab	Ab	Ab	Ab	Ab	Ab	After error flow detected
Ab	Ab	Ab	Ab	Ab	Ab	Ab	After error flow detected
Ab	Ab	Ab	Ab	Ab	Ab	Ab	After response from partner
Ab[35]	Ab[35]	2 or 5[33]	2 or 5[33]	2 or 5[33]	=	Ab[35]	After response from partner
Ab[35]		5	2	12	=	Ab[35]	After response from partner
	Ab[35]						
2 or 5[33]	2 or 5[33]	2 or 5[33]	2 or 5[33]	2 or 5[33]	=	2 or 5[33]	After rollback across LUW
Ab	Ab	Ab	Ab	Ab	Ab	Ab	Immediately
Ab	Ab	Ab	Ab	Ab	End	Ab	Immediately

Note:

[28] The back-end transaction starts in **receive state** after the front-end transaction has issued CONNECT PROCESS.

[29] No data may be included with SEND CONFIRM.

[30] CICS/400 does not support the CICS/ESA deviations from the LU6.2 architecture for SEND LAST WAIT, SEND LAST CONFIRM, and FREE.

[31] Where APPC transaction routing is taking place, the ISSUE SIGNAL command is invalid in this state.

[32] The commands SYNCPOINT and SYNCPOINT ROLLBACK do not relate to any particular conversation. They are propagated on all the conversations that are currently active for the task.

[33] The state of each conversation after rollback depends on several factors:

- The system you are communicating with. CICS handling of rollback varies with the family type and version; your partner system may handle it differently from CICS/400.
- The conversation state at the beginning of the current distributed unit of work. This state is the one adopted according to the APPC architecture. CICS/400 follows the architecture.

A conversation may be in **free state** after rollback if it has been terminated in one of these ways:

- Abnormally due to session failure or deallocate abend being received.
- Because the partner transaction hs issued a SEND LAST WAIT or FREE command.

After a syncpoint or rollback, it is advisable to determine the conversation state before issuing any further commands against the conversation.

[34] This results, not in an ATC abendk but in a INVREQ return code.

[35] This causes an ASP2 abend, not an ATCV.

[36] Although ISSUE PREPARE can return with the conversation in either **syncsend state**, **syncreceive state**, or **syncfree state**, the only commands allowed on that conversation following an ISSUE PREPARE are SYNCPOINT and SYNCPOINT ROLLBACK. All other commands abend ATCV.

Initial states

A front-end transaction in a conversation must issue an ALLOCATE command to acquire a session. If the session is successfully allocated, the front-end transaction's side of the conversation goes into **allocated state** (state 1).

A back-end transaction is initially in **receive state** (state 5).

Testing the conversation state

There are two ways for a transaction to inquire on the current state of one of its conversations.

The first is to use the EXEC CICS EXTRACT ATTRIBUTES STATE command and the second is to use the STATE parameter on the DTP commands. In both cases the current state is returned to the application in a CICS value data area (CVDA). Table 13 shows how the CVDA codes relate to the conversation state. The table also shows the symbolic names defined for these CVDA values.

Table 13. The conversation states

States used in this book		**States used in DTP programs**	
State name	**State number**	**Symbolic name**	**CVDA code**
Allocated	1	DFHVALUE(ALLOCATED)	81
Send	2	DFHVALUE(SEND)	90
Pendreceive	3	DFHVALUE(PENDRECEIVE)	87
Pendfree	4	DFHVALUE(PENDFREE)	86
Receive	5	DFHVALUE(RECEIVE)	88
Confreceive	6	DFHVALUE(CONFRECEIVE)	83
Confsend	7	DFHVALUE(CONFSEND)	84
Conffree	8	DFHVALUE(CONFFREE)	82
Synreceive	9	DFHVALUE(SYNCRECEIVE)	92
Syncsend	10	DFHVALUE(SYNCSEND)	93
Syncfree	11	DFHVALUE(SYNCFREE)	91
Free	12	DFHVALUE(FREE)	85
Rollback	13	DFHVALUE(ROLLBACK)	89

End of General-use programming interface

Part 4. Appendixes

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
500 Columbus Avenue
Thornwood, NY 10594
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 106-0032, Japan

The following paragraph does not apply to 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 information 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 Corporation
Software Interoperability Coordinator
3605 Highway 52 N

Rochester, MN 55901-7829
U.S.A.

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 information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Programming Interface Information

This publication is intended to help you to set up a CICS/400 system to communicate with another CICS/400 system or another CICS product.

This publication also documents General-Use Programming Interface and Associated Guidance Information.

General-Use programming interfaces allow the customer to write programs that obtain the services of CICS/400.

General-Use Programming Interface and Associated Guidance Information is identified where it occurs, either by an introductory statement to a chapter or section or by the following: **General-Use Programming Interface:**

Trademarks

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

Advanced Peer-to-Peer Networking
AIX
Application System/400
APPN
AS/400
CICS
CICS/400
DB2
e (Stylized)
IBM
IMS
iSeries
iSeries 400
Operating System/400
OS/2
OS/390
OS/400
pSeries
Redbooks
RISC System/6000
S/370
System/370
System/390
TXSeries
VSE/ESA

VTAM
Z/OS
zSeries
400

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

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

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

Index

D

E

F

G

I

J

L

M

N

O

P

Q

R

S

T

U

W

Readers' Comments — We'd Like to Hear from You

iSeries
CICS for iSeries Intercommunication
Version 5

Publication No. SC41-5456-00

Overall, how satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Overall satisfaction	☐	☐	☐	☐	☐

How satisfied are you that the information in this book is:

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Accurate	☐	☐	☐	☐	☐
Complete	☐	☐	☐	☐	☐
Easy to find	☐	☐	☐	☐	☐
Easy to understand	☐	☐	☐	☐	☐
Well organized	☐	☐	☐	☐	☐
Applicable to your tasks	☐	☐	☐	☐	☐

Please tell us how we can improve this book:

Thank you for your responses. May we contact you? ☐ Yes ☐ No

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.

Name

Address

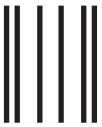
Company or Organization

Phone No.

Readers' Comments — We'd Like to Hear from You
SC41-5456-00

Cut or Fold
Along Line

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



NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM CORPORATION
ATTN DEPT 542 IDCLERK
3605 HWY 52 N
ROCHESTER MN 55901-7829

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

Cut or Fold
Along Line

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SC41-5456-00