CICS® Transaction Server for z/OS™

CICS Distributed Transaction Programming Guide

Version 2 Release 2

SC34-5998-00

CICS® Transaction Server for z/OS™

CICS Distributed Transaction Programming Guide

Version 2 Release 2

SC34-5998-00

Note!

Before using this information and the product it supports, be sure to read the general information under “Notices” on page 193.

Second edition (January 2002)

This edition applies to Version 2 Release 2 of CICS Transaction Server for z/OS, program number 5697-E93, and to all subsequent versions, releases, and modifications until otherwise indicated in new editions. Make sure you are using the correct edition for the level of the product.

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

At the back of this publication is a page entitled “Sending your comments to IBM”. If you wish to send comments by mail, please address them to:

User Technologies Department
Mail Point 095
IBM United Kingdom Laboratories
Hursley Park
WINCHESTER
Hampshire
SO21 2JN
United Kingdom

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

Contents

Preface . . . ix
What this book is about . . . ix
Who this book is for . . . ix
What is not covered by this book . . . ix
What you need to know to understand this book . . . ix
How to use this book . . . ix

Summary of changes . . . xi
Changes for CICS Transaction Server for z/OS, Version 2 Release 2 . . . xi
Changes for CICS Transaction Server for z/OS Version 2 Release 1 . . . xi
Changes for CICS Transaction Server for OS/390 Version 1 Release 3 . . . xi
Changes for CICS Transaction Server for OS/390 Version 1 Release 2 . . . xi
Changes for CICS Transaction Server for OS/390 Version 1 Release 1 . . . xi
Changes for CICS/ESA 4.1 . . . xi

Part 1. Concepts and design considerations . . . 1

Chapter 1. Concepts of distributed transaction processing (DTP) . . . 3
DTP's place in the CICS intercommunication facilities . . . 3
What is DTP? . . . 4
  Conversations. . . . 5
  Sessions . . . 6
Distributed processes . . . 7
Maintaining data integrity . . . 8
  Synchronization levels. . . . 9

Chapter 2. Designing distributed processes . . . 11
Structuring distributed transactions. . . . 11
  Avoiding performance problems . . . 11
  Making maintenance easier . . . 12
  Going for reliability . . . 12
  Protecting sensitive data . . . 12
  Maintaining connectivity . . . 12
  Safeguarding data integrity . . . 12
Designing conversations . . . 14
  Selecting the protocol . . . 15
APPC protocol . . . 16
  Selecting the APPC interface. . . . 16
  Selecting the APPC conversation type . . . 17
  Using VTAM persistent session support . . . 18
  Writing programs for APPC conversations . . . 19

Part 2. Writing programs for APPC mapped conversations . . . 21

Chapter 3. APPC mapped conversation flow . . . 23
Starting the conversation . . . 23
  Conversation initiation . . . 23
  Back-end transaction initiation . . . 25
  What happens if the back-end transaction fails to start . . . 26
Transferring data on the conversation . . . 27
  Sending data to the partner transaction . . . 27
  Switching from sending to receiving data . . . 28

Receiving data from the partner transaction 29
The CONVERSE command 31
Communicating errors across a conversation 32
Requesting INVITE from the partner transaction. 32
Demanding INVITE from the partner transaction. 32
Safeguarding data integrity 32
How to synchronize a conversation using CONFIRM commands 33
How to synchronize conversations using SYNCPOINT commands 35
Ending the conversation 35
Normal termination of a conversation. 35
Emergency termination of a conversation 36
Unexpected termination of a conversation 36
Checking the outcome of a DTP command 37
Checking EIB fields and the conversation state 39
Summary of CICS commands for APPC mapped conversations 40

Chapter 4. State transitions in APPC mapped conversations 41
The state tables for APPC mapped conversations 41
How to use the state tables 41
Initial states 50
Testing the conversation state 50

Part 3. Writing programs for MRO conversations 51

Chapter 5. MRO conversation flow 53
Starting the conversation 53
Conversation initiation 53
Back-end transaction initiation 55
Transferring data on the conversation 56
Sending data to the partner transaction 56
Switching from sending to receiving data 57
Receiving data from the partner transaction 58
The CONVERSE command 60
Safeguarding data integrity 60
Ending the conversation 60
Ending a conversation normally. 60
Unexpected termination of a conversation 60
Checking the outcome of a DTP command 61
Checking EIB fields and the conversation state 62
Summary of commands for MRO conversations. 63

Chapter 6. State transitions in MRO conversations 65
The state table for MRO conversations 65
How to use the state table. 65
Initial states 68
Testing the conversation state 68

Part 4. Writing programs for APPC basic conversations 69

Chapter 7. APPC basic conversation flow 71
Starting the conversation 71
Conversation initiation 71
Back-end transaction initiation 73
What happens if the back-end transaction fails to start up 74
Sending data to the partner transaction 75

Switching from sending to receiving data 76
Receiving data from the partner transaction 77
Receiving data by the record. 78
Receiving data by the buffer 79
Communicating errors across a conversation 80
Requesting INVITE from the partner transaction. 80
Demanding INVITE from the partner transaction. 80
Safeguarding data integrity 81
How to synchronize conversations using CONFIRM commands 81
How to synchronize conversations using SYNCPOINT commands 83
Ending the conversation 83
Normal termination of a conversation. 83
Emergency termination of a conversation 84
Unexpected termination of a conversation 84
Checking the outcome of GDS commands. 85
Testing for request failure 85
Testing indicators 86
Checking CONVDATA fields and the conversation state 89
Summary of commands for APPC basic conversations 91

Chapter 8. State transitions in APPC basic conversations. 93
The state tables for APPC basic conversations 93
How to use the state tables 93
Initial states 102
Testing the conversation state 102

Part 5. Writing programs for LUTYPE6.1 conversations 103

Chapter 9. LUTYPE6.1 conversation flow 105
Starting the conversation. 105
Conversation initiation 105
Back-end transaction initiation 106
Transferring data on the conversation 106
Sending data to the partner transaction 106
Switching from sending to receiving data 106
Receiving data from the partner transaction 107
Waiting for a signal 107
Combining sending and receiving 107
Communicating errors across a conversation 107
Safeguarding data integrity 107
Ending the conversation 108
Ending a conversation normally 108
Unexpected termination of a conversation 108
Checking the outcome of a DTP command 108
Considerations for the front-end transaction 110
Session allocation 110
The session identifier 111
Summary of commands for LUTYPE6.1 conversations 112

Chapter 10. State transitions in LUTYPE6.1 conversations 113
The state table for LUTYPE6.1 conversations 113
How to use the state table 113
Initial states. 116
Testing the conversation state 116

Part 6. Syncpointing a distributed process 117

Chapter 11. Syncpointing a distributed process 119
The SYNCPOINT command 119
The ISSUE PREPARE command 120
The SYNCPOINT ROLLBACK command 120
When a backout is required 121
Synchronizing two CICS systems 121
SYNCPOINT in response to SYNCPOINT 121
SYNCPOINT in response to ISSUE PREPARE 124
SYNCPOINT ROLLBACK in response to SYNCPOINT ROLLBACK 125
SYNCPOINT ROLLBACK in response to SYNCPOINT 126
SYNCPOINT ROLLBACK in response to ISSUE PREPARE 127
ISSUE ERROR in response to SYNCPOINT 128
ISSUE ERROR in response to ISSUE PREPARE 129
ISSUE ABEND in response to SYNCPOINT 130
ISSUE ABEND in response to ISSUE PREPARE 131
Session failure in response to SYNCPOINT 132
Session failure in response to ISSUE PREPARE 134
Session failure in response to SYNCPOINT ROLLBACK 135
Synchronizing three or more CICS systems 136
SYNCPOINT in response to SYNCPOINT 136
SYNCPOINT ROLLBACK in response to SYNCPOINT 139
Session failure and the indoubt period 141
What really flows between APPC systems 141

Part 7. Appendixes 145

Appendix A. CICS mapping to the APPC architecture 147
Command mapping for APPC basic conversations 148
Return codes for APPC basic conversations 153
Command mapping for APPC mapped conversations 155
Return codes for APPC mapped conversations 160
CICS deviations from the APPC architecture 161
Effects of CICS deviations on the transaction programmer 162

Appendix B. Migration of LUTYPE6.1 applications to APPC links 165
Migration mode 165
State transitions in LUTYPE6.1 migration-mode conversations 167

Appendix C. Differences between APPC mapped and MRO conversations 173
Different treatment of command sequences 173
Using the LAST option 174
The LAST option and syncpoint flows on APPC sessions 174
The LAST option and syncpoint flows on MRO sessions 174

Appendix D. Below the SNA interface 175
SNA indicators and records 175
Request mode and responses 176
When SNA indicators are transmitted 176

Glossary 177

Bibliography 181
CICS Transaction Server for z/OS 181

CICS books for CICS Transaction Server for z/OS 181
CICSPlex SM books for CICS Transaction Server for z/OS 182
Other CICS books 182
Books from related libraries 182
IMS 182
Systems Application Architecture (SAA) 183
Systems Network Architecture (SNA) 183
Determining if a publication is current 183

Accessibility 185

Index 187

Notices 193
Programming Interface Information 194
Trademarks. 194

Sending your comments to IBM 195

Preface

What this book is about

This book discusses the technique (called **distributed transaction processing** or DTP) of spreading the functions of a transaction over several transaction programs within a network. The book also provides guidance in producing application programs that exchange data through distributed transaction processing (DTP) on Advanced Program-to-Program Communication (APPC), multiregion operation (MRO), and LUTYPE6.1 links.

Who this book is for

This book is for anyone who is involved in systems design and programming for CICS® DTP applications.

What is not covered by this book

This book discusses only distributed transaction processing. The other basic intercommunication facilities provided by CICS are described in the *CICS Intercommunication Guide*.

Methods of accessing CICS programs and transactions from non-CICS environments are described in the *CICS External Interfaces Guide*.

CICS Transaction Server for z/OS's support for the CICS Client family of workstation products is described in the *CICS Family: Communicating from CICS on System/390*.

What you need to know to understand this book

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

Readers will find it easier to understand the concepts discussed in this book if they have read Part 1 of the *CICS Intercommunication Guide*.

How to use this book

"Part 1. Concepts and design considerations" on page 1 is a very important framework within which the rest of the book can be understood. You should therefore start by reading this section to familiarize yourself with the concepts of DTP and the things you have to think about when designing such applications.

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

Summary of changes

This book is based on the CICS Distributed Transaction Programming Guide for CICS Transaction Server for z/OS Version 2 Release 1, SC34-5708-00. Changes from that edition are marked by vertical bars in the left margin.

This part lists briefly the changes that have been made for the following recent releases:

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

There are no significant changes for this edition.

Changes for CICS Transaction Server for z/OS Version 2 Release 1

There were no significant changes for this edition.

Changes for CICS Transaction Server for OS/390 Version 1 Release 3

There were no significant changes for this edition.

Changes for CICS Transaction Server for OS/390 Version 1 Release 2

There were no significant changes for this edition.

Changes for CICS Transaction Server for OS/390 Version 1 Release 1

There were no significant changes for this edition.

Changes for CICS/ESA 4.1

The main changes made for this edition were:

- The effects of using VTAM persistent sessions were described in "Using VTAM persistent session support" on page 18.
- CICS/ESA 4.1 support for the PARTNER option on EXEC CICS ALLOCATE, CONNECT PROCESS, GDS ALLOCATE, and GDS CONNECT PROCESS commands was reflected in changes to the chapters that describe how to write APPC mapped and basic programs.

Part 1. Concepts and design considerations

This part of the book describes the basic concepts of CICS distributed transaction processing (DTP) and what you must consider when designing DTP applications.

"Chapter 1. Concepts of distributed transaction processing (DTP)" on page 3 defines DTP and discusses how conversation partners can work together.

"Chapter 2. Designing distributed processes" on page 11 discusses the issues you must think about in designing a DTP application.

Chapter 1. Concepts of distributed transaction processing (DTP)

This chapter explains what distributed transaction processing (DTP) is. It contains the following topics:

- DTP’s place in the CICS intercommunication facilities
- “What is DTP?” on page 4
- “Distributed processes” on page 7
- “Maintaining data integrity” on page 8.

DTP’s place in the CICS intercommunication facilities

Today, an increasing number of organizations are connecting their information systems together and distributing resources among them. To support this kind of processing, applications need to be designed and developed to access resources across multiple systems. So CICS provides the following basic intercommunication facilities:

- **Function shipping**, which enables your application program to access resources in another CICS system.
- **Distributed program link**, which enables a program in one CICS system to issue a link command that invokes a program in another CICS system, waiting for a RETURN.
- **Asynchronous processing**, which enables a CICS transaction to initiate a transaction in another CICS system and pass data to it.
- **Transaction routing**, which enables a terminal connected to one CICS system to run a transaction in another CICS system.
- **Distributed transaction processing**, which enables a CICS transaction to communicate with a transaction running in another system. The transactions are designed and coded specifically to communicate with each other, and in doing so to use the intersystem link with maximum efficiency.

In addition, CICS provides the following methods of accessing CICS programs and transactions from non-CICS environments:

- The CICS bridge
- The external CICS interface (EXCI)
- Transactional EXCI
- Support for DCE Remote Procedure Calls
- Support for ONC Remote Procedure Calls
- Inter-orb Protocol (IIOP)
- The Web interface.

This book discusses only distributed transaction processing. The other basic intercommunication facilities are described in the *CICS Intercommunication Guide*. Methods of accessing CICS programs and transactions from non-CICS environments are described in the *CICS External Interfaces Guide* and the *CICS Internet Guide*.

What is DTP?

DTP is one of the ways in which CICS allows processing to be split between intercommunicating systems. Only DTP allows two or more communicating application programs to run simultaneously in different systems and to pass data back and forth between themselves—that is, to carry on a conversation.

Of the intercommunication facilities offered by CICS, DTP is the most flexible and powerful, but also the most complex. This chapter introduces you to the basic concepts involved in creating DTP applications. For a broad discussion of intercommunication concepts, see *CICS Intercommunication Guide*.

DTP allows two or more partner programs in different systems to interact with each other for some purpose. DTP enables a CICS transaction to communicate with one or more transactions running in different systems. A group of such connected transactions is called a **distributed process**.

The process can best be shown by discussing the operation of DTP between two CICS systems, CICSA and CICSB. The configuration is shown in Figure 1.

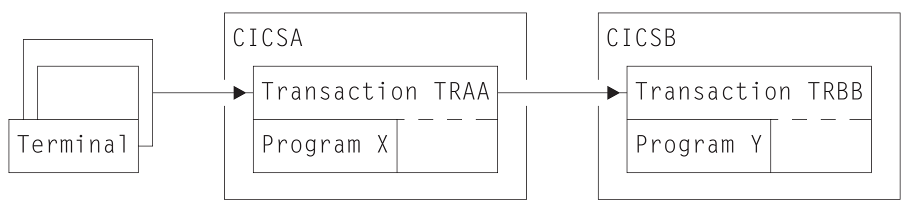

Figure 1. DTP between two CICS transactions

1. A transaction (TRAA) is initiated on CICSA, for example, by a terminal operator keying in a transaction ID and initial data.
2. To fulfill the request, the processing program X begins to execute on CICSA, probably reading initial data from files, perhaps updating other files and writing to print queues.
3. Without ending, program X asks CICSA to establish a communication session with another CICS system, CICSB. CICSA responds to the request.
4. Also without ending, program X sends a message across the communication session, asking CICSB to start a new transaction, TRBB. CICSB initiates transaction TRBB by invoking program Y.
5. Program X now sends and receives messages, including data, to and from program Y. Between sending and receiving messages, both program X and program Y continue normal processing completely independently. When the two programs communicate, their messages can consist of:
   - Agreements on how to proceed with communication or how to end it. For example, program X can tell program Y when it may transmit messages across the session. At any time, both programs must know the state of their communication, and thus, what actions are allowed. At any time, either system may have actual control of the communication.
   - Agreements to make permanent all changes made up to that point. This allows the two programs to **synchronize** changes. For example, a dispatch billing program on CICSA might wish to commit delivery and charging for a

stock item, but only when a warehouse program in CICSB confirms that it has successfully allocated the stock item and adjusted the inventory file accordingly.

- Agreements between CICSA and CICSB to cancel, rather than make permanent, changes to data made since a given point. Such a cancelation (or rollback) might occur when customers change their minds, for example. Alternatively, it might occur because of uncertainty caused by failure of the application, the system, the communication path, or the data source.

Although the two programs X and Y exist as independent units, it is clear that they are designed to work as one. Of course, DTP is not limited to pairs of programs. You can chain many programs together to distribute processing more widely. This is discussed later in the book.

In the overview of the process given above, the location of program Y has not been specified. Program X is a CICS program, but program Y need not be, because CICS can establish sessions with non-CICS, LUTYPE6.1, MRO, or APPC partners. This is discussed in “Chapter 2. Designing distributed processes” on page 11.

The rest of this book considers the cases of CICS DTP with the following protocols:
- APPC mapped
- MRO
- APPC basic
- LUTYPE6.1.

Conversations

Although several programs can be involved in a single distributed process, information transfer within the process is always between self-contained **communication pairs**. The exchange of information between a pair of programs is called a **conversation**. During a conversation, both programs are active; they send data to and receive data from each other. The conversation is two-sided but at any moment, each partner in the conversation has more or less control than the other. According to its level of control (known as its **conversation state**), a program has more or less choice in the commands that it can issue.

Conversation states

Thirteen conversation states have been defined for CICS DTP. The set of states possible for a particular conversation depends on the protocol and synchronization level used. (The concepts of protocol and synchronization level are explained in “Selecting the protocol” on page 15 and “Maintaining data integrity” on page 8 respectively.) Table 1 on page 6 shows which conversation states are defined for which protocols and synchronization levels.

Table 1. The conversation states defined for different protocols. Yes and no indicate whether the state is defined.

State number	State name	APPC sync level 0	APPC sync level 1	APPC sync level 2	MRO	LUTYPE6.1 normal mode	LUTYPE6.1 migration mode
1	Allocated	Yes	Yes	Yes	Yes	Yes	Yes
2	Send	Yes	Yes	Yes	Yes	Yes	Yes
3	Pendreceive	Yes	Yes	Yes	No	Yes	Yes
4	Pendfree	Yes	Yes	Yes	Yes	Yes	Yes
5	Receive	Yes	Yes	Yes	Yes	Yes	Yes
6	Confreceive	No	Yes	Yes	No	No	Yes
7	Confsend	No	Yes	Yes	No	No	Yes
8	Conffree	No	Yes	Yes	No	No	Yes
9	Syncreceive	No	No	Yes	Yes	Yes	Yes
10	Syncsend	No	No	Yes	No	Yes	Yes
11	Syncfree	No	No	Yes	Yes	Yes	Yes
12	Free	Yes	Yes	Yes	Yes	Yes	Yes
13	Rollback	No	No	Yes	Yes	No	Yes

By using a special CICS command (EXTRACT ATTRIBUTES STATE), or the STATE option on a conversation command, a program can obtain a value that indicates its own conversation state. CICS places such a value in a variable named by the program; the variable is sometimes referred to as a **state variable**. Knowing the current conversation state, the program then knows which commands are allowed. If, for example, a conversation is in **send state**, the transaction can send data to the partner. (The transaction can take other actions instead, as indicated in the relevant state table.)

When a transaction issues a DTP command, this can cause the conversation state to change. For example, a transaction can deliberately switch the conversation from **send state** to **receive state** by issuing a command that invites the partner to send data. When a conversation changes from one state to another, it is said to undergo a **state transition**. The state tables in later chapters show how these transitions take place.

Not only does the conversation state determine what commands are allowed, but the state on one side of the conversation reflects the state on the other side. For example, if one side is in **send state**, the other side is in either **receive state**, **confreceive state**, or **syncreceive state**.

Sessions

A conversation takes place across a CICS resource called a **session**. One transaction (known as the **front-end transaction**) asks CICS to allocate a session, and then uses this session to request that the remote transaction (known as the **back-end transaction**) be initiated. Then the two transactions, which can be thought of as partners in the conversation, can “talk to” each other.

A session is a logical data path between two logical units. It is a shared resource and is allocated to a transaction in response to a request from the transaction.

Resource definition determines the number of sessions available for allocation. While a conversation is active, it has sole use of the session allocated to it.

A transaction starts a conversation by requesting the use of a session to a remote system. When it obtains the session, the transaction can issue commands that cause an *attach* request to be sent to the other system to activate the transaction that is to be the conversation partner. A transaction can issue an attach request to more than one other transaction.

Distributed processes

A transaction can initiate other transactions, and hence, conversations. In a complex process, a distinct hierarchy emerges, usually with the terminal-initiated transaction at the top. Figure 2 shows a possible configuration. In this example, transaction TRAA, in system CICSA, is initiated from a terminal. Transaction TRAA attaches transaction TRBB to run in system CICSB. Transaction TRBB in turn attaches transaction TRCC in system CICSC and transaction TRDD in system CICSD. Both transactions TRCC and TRDD attach the same transaction SUBR in system CICSE, thus giving rise to two copies of SUBR.

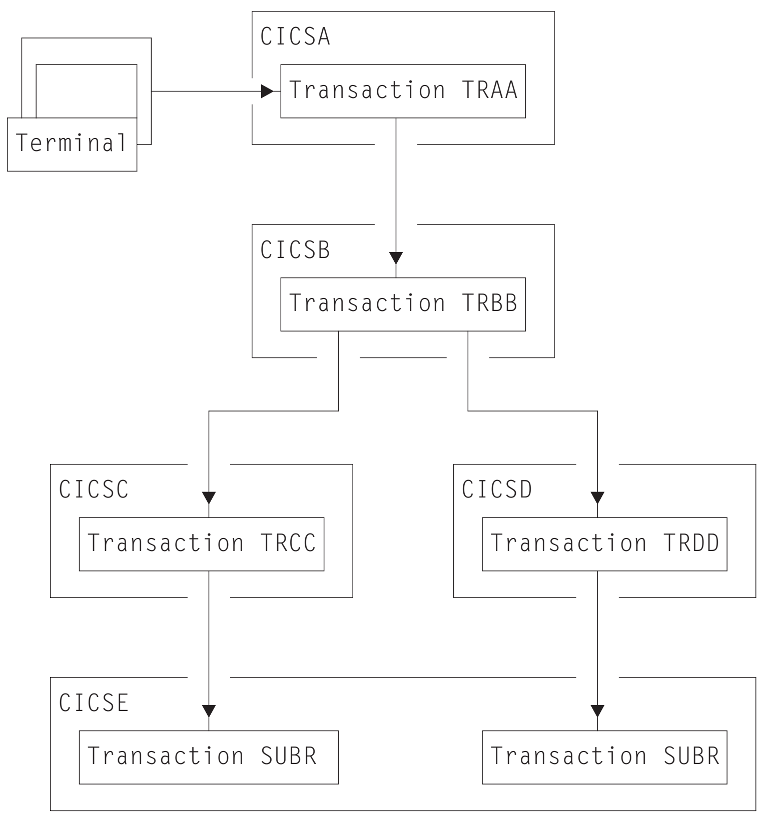

Figure 2. DTP in a distributed process. (Arrowheads indicate attach requests).

Notice that, for every transaction, there is only one *inbound* attach request, but that there can be a number of *outbound* attach requests. The session that activates a transaction is called its **principal facility**. A session that is allocated by a transaction to activate another transaction is called its **alternate facility**. Therefore, a transaction can have only one principal facility, but several alternate facilities.

When a transaction initiates a conversation, it is the front-end transaction on that conversation. Its conversation partner is the back-end transaction on the same conversation. It is normally the front-end transaction that dominates, and determines the way the conversation goes. This style of processing is sometimes referred to as the client/server model. (In some books, it is called master/slave.)

Alternatively, the front-end transaction and back-end transaction may switch control between themselves. This style of processing is called **peer-to-peer**. As the name implies, this model describes communication between equals. You are free to select whichever model you need when designing your application; CICS supports both.

Maintaining data integrity

You should design your application to cope with the things that can go wrong while a transaction is running, for example, a session failing. The conversation protocol helps you recover from errors and ensures that the two sides remain in step with each other. This use of the protocol is called **synchronization**.

Synchronization allows you to protect recoverable resources such as transient data queues and files, whether they are local or remote. *Whatever goes wrong during the running of a transaction should not leave the associated resources in an inconsistent state*.

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

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

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

When a transaction issues a syncpoint command, CICS **commits** all changes to recoverable resources associated with that transaction. After the syncpoint, the transaction can no longer back out changes made since the previous syncpoint. They have become irreversible.

Although CICS can commit and backout changes to local and remote resources for you, this service must be paid for in performance. If the recovery of resources throughout a distributed process is not a problem (for example, in an inquiry-only application), you can use simpler methods of synchronization.

Synchronization levels

Systems Network Architecture (SNA) defines three levels of synchronization for conversation using the APPC protocol:

- Level 0 – None
- Level 1 – Confirm
- Level 2 – Syncpoint[1] .

At sync level 0, there is no CICS support for synchronization of remote resources on connected systems. But it is still possible, under the control of the application to achieve some degree of synchronization by interchanging data, using the SEND and RECEIVE commands.

At sync level 1, you can use special commands for communication between the two conversation partners. One transaction can *confirm* the continued presence and readiness of the other. Both transactions are responsible for preserving the data integrity of recoverable resources by issuing syncpoint requests at the appropriate times.

At sync level 2, all syncpoint requests are automatically propagated across multiple systems. CICS implies a syncpoint when it starts a transaction; that is, it initiates logging of changes to recoverable resources, but no control flows take place. CICS takes a syncpoint when one of the transactions terminates normally. One abending transaction causes all to rollback. The transactions themselves can initiate syncpoint or rollback requests. However, a syncpoint or rollback request is propagated to another transaction only when the originating transaction is in conversation with the other transaction, and sync level 2 has been selected.

Bear in mind that syncpoint and rollback are not limited to any one conversation within a transaction. They are propagated on every conversation currently active at sync level 2.

1. Sync level 2 is not supported on single-session connections.

Chapter 2. Designing distributed processes

This chapter discusses the issues you must consider when designing distributed processes to run under APPC or MRO. These issues include structuring distributed processes and designing conversations.

It is assumed that you are already familiar with the issues involved in designing applications in single CICS systems, as described in the *CICS Application Programming Guide*.

The chapter contains the following topics:
- Structuring distributed transactions
- "Designing conversations" on page 14
- "APPC protocol" on page 16.

Structuring distributed transactions

As with many design problems, designing a DTP application involves dealing with several conflicting objectives that must be carefully balanced against each other. These include performance, ease of maintenance, reliability, security, connectivity to existing functions, and recovery.

Avoiding performance problems

If performance is the highest priority, you should design your application so that data is processed as close to its source as possible. This avoids unnecessary transmission of data across the network. Alternatively, if processing can be deferred, you may wish to consider batching data locally before transmitting.

To maintain performance across the intersystem connection, the conversation should be freed as soon as possible — so that the session may be used by other transactions. In particular, avoid holding a conversation across a terminal wait.

In terminal-attached transactions, pseudo-conversational design improves performance by reducing the amount of time a transaction holds CICS resources. This is because a terminal user is likely to take seconds or even minutes to respond to any request for keyboard input. In contrast, the communication delay associated with a conversation between partner transactions is likely to be only a few milliseconds. It is therefore not necessary to terminate a front-end transaction pending a response from a back-end transaction.

However, a front-end transaction can be terminal-initiated, in which case a pseudo-conversational design may be appropriate. When input from the terminal user is required, the front-end transaction and its conversations should be terminated. After the terminal user has responded, the successor front-end transaction can initiate a successor back-end transaction. If the first back-end transaction needs to pass information to its successor, the information must either be passed to the front-end transaction or stored locally (for example, in temporary storage).

Stored information should be retrievable by identifiers that are not associated with the particular session used by the conversation. The back-end transaction cannot use a COMMAREA, a RETURN TRANSID, nor a TCTUA for this purpose. Instead, it can construct the identifier of a temporary-storage queue by using information obtained from the front-end transaction. The sysid of the principal facility, and the identifier of the terminal to which the front-end transaction is attached, can be used.

Making maintenance easier

To correct errors or to adapt to the evolving needs of an organization, distributed processes inevitably need to be modified. Whether these changes are made by the original developers or by others, this task is likely to be easier if the distributed processes are relatively simple. So consider minimizing the number of transactions involved in a distributed process.

Going for reliability

If you are particularly concerned with reliability, consider minimizing the number of transactions in the distributed process.

Protecting sensitive data

If the distributed process is to handle security-sensitive data, you could place this data on a single system. This means that only one of the transactions needs knowledge of how or where the sensitive data is stored. For guidance on implementing security in CICS systems, see the *CICS RACF Security Guide*.

Maintaining connectivity

If you require connectivity to transactions running in a back-level CICS system, check the appropriate books for that release to ensure that the functions required are compatible.

The following aspects of distributed process design differ from single-system considerations:

Data conversion
For non-EBCDIC APPC logical units (for example, CICS OS/2), some data conversion may be required on either receipt or sending of data.

Using multiple conversations
When using multiple, serial conversations, note that different conversation identifiers may be provided to the transaction (by CICS). It is therefore not advisable to use the conversation identifier for naming resources (for example, temporary storage queues).

Safeguarding data integrity

If it is important for you to be able to recover your data when things go wrong, design conversations for sync level 2, and keep the units of work as small as possible. However, this is not always possible, because the size of a UOW is determined largely by the function being performed. Remember that CICS syncpoint processing has no information about the structure and purpose of your application. As an application designer, you must ensure that syncpoints are taken at the right time and place, and to good purpose. If you do, error conditions are unlikely to lead to inconsistencies in recoverable data resources.

Here is an example of a distributed application that transfers the contents of a temporary storage queue from system A to system B, using a pair of transactions (TRAA in system A, and TRBB in system B), and a conversation at synclevel 2.

1. Transaction TRAA in system A reads a record from the temporary storage queue.
2. Transaction TRAA sends the record to system B, and waits for the response.
3. Transaction TRBB in system B receives the record from system A.
4. Transaction TRBB processes the record, and sends a response to system A.
5. Transaction TRAA receives the response, and deletes the record from the temporary storage queue.

These steps are repeated as long as there are records remaining in the queue. When the queue is empty:

6. Transaction TRAA sends a 'last record' indicator to system B.
7. Transaction TRBB sends a response to system A.

There are several points at which you may consider taking a syncpoint. Here are the relative merits of taking a syncpoint at each of these points:

At the start of processing
Because a UOW starts at this point, a syncpoint has no effect. In fact, if TRBB tries to take a syncpoint without having first issued a command to receive data, it will be abended.

After transaction TRAA receives a response
A syncpoint at this point causes CICS to commit a record in system B before it has been deleted from system A. If either system (or the connection between them) fails before the distributed process is completed, data may be duplicated.

Immediately after the record is deleted from the temporary storage queue
Because minimum processing is needed before resources are committed, this may be a safe place to take a syncpoint if the queue is long or the records are large. However, performance may be poor because a syncpoint is taken for each record transmitted.

After transaction TRAA receives the response to the last-record indicator
If you take a syncpoint only when all records have been transmitted, an earlier failure will mean that all data will have to be retransmitted. A distributed process that syncpoints only at this stage will complete more quickly than one that syncpoints after each record is processed, provided no failure occurs. However, it will take longer to recover. If more than two systems are involved in the process, this problem is made worse.

Bear in mind that too many conversations within one distributed transaction complicates error recovery. A complex structure may sometimes be unavoidable, but usually it means that the design could be improved if some thought is given to simplifying the structure of the distributed transaction.

A UOW must be recoverable for the whole process of which it forms a part. All changes made by both partners in every conversation must be backed out if the UOW does not complete successfully. Syncpoints are not arbitrary divisions, but must reflect the functions of the application. Units of work must be designed to preserve consistent resources so that when a transaction fails, **all** resources are restored to their correct state.

Before terminating a sync level-2 conversation, make sure that the partner transaction is able to communicate any errors that it may have found. Not doing so may jeopardize data integrity.

Designing conversations

Once the overall structure of the distributed process has been decided, you can then start to design individual conversations. Designing a conversation involves deciding what functions to put into the front-end transaction and into the back-end transaction, and deciding what should be in a distributed unit of work. So you have to make decisions about how to subdivide the work to be done for your application.

Because a conversation involves transferring data between two transactions, to function correctly, each transaction must know what the other intends. For instance, there is little point in the front-end transaction sending data if all the back-end transaction is designed to do is print the weekly sales report. You must therefore 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 single conversation and not to a transaction as a whole. In each state, there are a number of commands that might reasonably be issued. The command itself, together with its outcome, may cause the conversation to change from one state to another.

To determine the conversation state, you can use either the STATE option on a command or the EXTRACT ATTRIBUTES STATE command. Note, however, that the STATE option is valid only for MRO and APPC sessions, not for LUTYPE6.1 sessions. For programming information about the state values returned by different commands, see *CICS Application Programming Reference*.

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 protocol, sync level, application programming interface (API), and conversation type that you choose. (Only the APPC protocol gives you a choice of APIs and conversation types.)

“Maintaining data integrity” on page 8 contains guidance on selecting the sync level for a conversation. “Chapter 11. Syncpointing a distributed process” on page 119 discusses the synchronization commands and their effects.

The following sections discuss how you choose the protocol, the API, and the conversation type. These sections also tell you where to find the state tables and command descriptions relevant to the choice you have made.

Selecting the protocol

CICS provides three different protocols:

- **APPC** (advanced program-to-program communication, sometimes referred to as LUTYPE6.2)
- **MRO** (multiregion operation)
- **LUTYPE6.1** (logical unit type 6.1).

These protocols define the rules under which two transactions can communicate with each other.

Both APPC and LUTYPE6.1 are protocols defined by SNA. They are therefore more widely available for communicating with non-CICS systems. LUTYPE6.1 is the predecessor of APPC; so you should, if possible, avoid using LUTYPE6.1 for new applications. However, some new applications may still need to use LUTYPE6.1 to communicate with existing LUTYPE6.1 applications.

To help you migrate applications from LUTYPE6.1 to APPC, CICS provides a migration path. For more information on this, see “Appendix B. Migration of LUTYPE6.1 applications to APPC links” on page 165.

Choosing between MRO and APPC can be quite simple. The options depend on the configuration of your CICS complex and on the nature of the conversation partner. MRO does not support communication with a partner in a non-CICS system. Further, it supports communication between transactions running in CICS systems in different MVS images only if the MVS images are in the same MVS sysplex, and are joined by cross-system coupling facility (XCF) links; the MVS images must be at IBM MVS/ESA™ release level 5.1, or later. (For full details of the hardware and software requirements for XCF/MRO, see the *CICS Intercommunication Guide*.)

For communication with a partner in another CICS system, where the CICS systems are either in the same MVS image, or in the same MVS/ESA 5.1 (or later) sysplex, you can use either the MRO or the APPC protocol. There are good performance reasons for using MRO. But if there is any possibility that the distributed transactions will need to communicate with partners in other operating systems, it is better to use APPC so that the transaction remains unchanged.

APPC application programs will not run under MRO. Even if both partners are in the same MVS image, CICS will not use MRO facilities but will send conversation data through the communications controller. That involves some VTAM overhead. So you must decide whether your application programs are to converse using APPC or MRO and code them accordingly.

Table 2 on page 16 points out the main differences between the MRO and APPC protocols.

Table 2. MRO protocol compared with APPC protocol

MRO	APPC
Function is realized without using a telecommunication access method.	Depends on VTAM® or similar.
Non-standard architecture.	SNA architecture.
CICS-to-CICS links only.	Links to non-CICS systems possible.
Communicates within single MVS image, or (using XCF/MRO) between MVS images in same sysplex.	Communicates across multiple MVS images or other operating systems.
Sync level 2 forced for the conversation.	Sync level 0, 1, or 2 can be selected.
Program initialization parameter (PIP) data not supported.	PIP data supported.
Data transmission not deferred.	Deferred data transmission.
Partner transaction may be identified in data.	Partner transaction defined by program command.
Performance overhead over a single application.	Even greater performance overhead over a single application.
RECEIVE can be issued only in receive state.	RECEIVE causes conversation turnaround when issued in send state on mapped conversations.
No ISSUE SIGNAL command.	ISSUE SIGNAL command available.
WAIT command has no function.	WAIT command causes transmission of deferred data.

If you decide to use the APPC protocol, see the next section APPC protocol and decide on which programming interface and which conversation type to use.

If you decide to use the MRO protocol, see “Part 3. Writing programs for MRO conversations” on page 51.

If you decide to use the LUTYPE6.1 protocol, see “Part 5. Writing programs for LUTYPE6.1 conversations” on page 103.

APPC protocol

If you choose to use APPC, you must decide which application programming interface (API) to use; and then which conversation type (basic or mapped) to use. See the following sections.

Selecting the APPC interface

CICS Transaction Server for z/OS, Version 2 Release 2 provides a choice of two application programming interfaces (APIs) for coding your DTP conversations on APPC sessions:

- **CICS API**, is the programming interface of the CICS implementation of the APPC architecture. It consists of EXEC CICS commands.
- **Common Programming Interface Communications** (CPI Communications) is the communications interface defined by the IBM Systems Application Architecture (SAA). It consists of a set of defined verbs in the form of program calls, which are adapted for the language being used.

As an existing CICS user, you should not need to convert to the CPI Communications interface unless you have decided to adopt it as standard. You should continue to use EXEC CICS. However, to help you review the choices, Table 3 makes a general comparison between the two methods.

Table 3. The CICS API compared with the CPI Communications interface

CICS API	CPI Communications interface
Portability between different members of the CICS family.	Portability between systems that support SAA.
Basic conversations can be programmed only in assembler and C language.	Basic conversations can be programmed in any of the available SAA languages.
Sync levels 0, 1, and 2 supported.	Sync levels 0, 1, and 2 supported, *except for transaction routing, for which only sync levels 0 and 1 are supported*.
PIP data supported.	PIP data not supported.
Can be used on the principal facility to a transaction started by automatic transaction initiation (ATI).	Cannot be used on the principal facility to a transaction started by ATI.
Limited compatibility with MRO.	No compatibility with MRO.
Commands similar to those used to communicate with IBM 3270 terminals.	Commands similar to those used to define the APPC architecture.
All parameters are passed on the relevant command.	Parameter values are set by special commands before the relevant command is issued.

For further information about CPI Communications, see the *Common Programming Interface Communications* manual, SC26-4399.

It is possible to mix CPI Communications calls and EXEC CICS commands in the same program, but not on the same side of the same conversation. It *is* possible to implement a distributed transaction where one partner to a conversation uses CPI Communications calls and the other uses the CICS API. To do this you have to know how the APIs on both sides map to the APPC architecture. See “Appendix A. CICS mapping to the APPC architecture” on page 147.

Selecting the APPC conversation type

The communication commands you code in your application depend on whether you intend to use **basic** or **mapped** conversations. CICS-to-CICS applications need use only mapped conversations. Basic conversations (also referred to as “unmapped”) are useful only when communicating with systems that do not support mapped conversations. These include some APPC devices.

The two conversation types are similar. The main difference lies in the way user data is formatted for transmission:

- In mapped conversations, the application merely sends the data to the partner.
- In basic conversations, the application has to add a few control bytes to convert the data into an SNA-defined format called a **generalized data stream** (GDS).

The CICS API uses the EXEC CICS GDS commands for basic conversations and terminal control type EXEC CICS commands for mapped conversations.

Table 4 summarizes the differences between mapped and basic conversations. Note that it only applies to the CICS API.

Table 4. APPC conversations – mapped compared with basic

Mapped	Basic
The conversation partners exchange only data that is relevant to the application.	Both partners must package the user data in GDS records before sending and unpack it on receipt.
All commands use the EXEC Interface Block for status reporting.	All commands use a RETCODE and CONVDATA for status reporting.
The transaction can *handle* exception conditions or let them default.	The transaction must test for exceptional conditions in a RETCODE.
A RECEIVE command issued in send state causes conversation turnaround.	A RECEIVE command is illegal in send state.
Transactions may be written in COBOL, PL/I, C, or assembler.	Transactions may be written in C or assembler.
By specifying the RTIMOUT option of the PROFILE definition, you can cause a conversation to time out if the partner does not respond.	You cannot cause a conversation to time out if the partner does not respond.

Using VTAM persistent session support

If you use VTAM persistent session support, after a CICS failure APPC sessions are held in “recovery pending” state until CICS restarts, or until the timeout value set on the PSDINT system initialization parameter expires.

If you enable persistent session support in the local CICS, DTP applications that use APPC sessions defined as persistent are affected as follows:

- Remote partner programs can cause excessive queuing delays in the partner system if they continue to issue commands on persistent APPC sessions after this CICS has failed. There is no way for the partner to know that a persistent sessions restart is in progress. However, there are various actions you can take to reduce the risk of new work building up for a connection to a persisting CICS Transaction Server for z/OS, Version 2 Release 2 system.

  Actions on the partner system:

  – In DTP applications, requests for sessions are instigated by EXEC CICS ALLOCATE commands. Control the overall number of queued session requests by using:

    - The QUEUELIMIT and MAXQTIME options on the CONNECTION definition
    - An XZIQUE global user exit program.

    These methods are described in the *CICS Intercommunication Guide*.

  – Control individual session requests by coding the NOQUEUE|NOSUSPEND option on EXEC CICS ALLOCATE commands.

  – Force mapped APPC RECEIVE or CONVERSE commands to time out if there is any delay in receiving expected data, by coding the RTIMOUT option on PROFILE definitions.

  Action on this system:

  – Code a PSDINT value that takes into account the number of your APPC sessions to partner systems.

- After a restart, LU6.2 session names, in the range −AAA to −999, are allocated on a “first free” basis (rather than on a “next in the sequence” followed by “last free” basis). This may affect applications that use LU6.2 CONVIDs as external qualifiers.

For further information about VTAM persistent session support, see the *CICS Recovery and Restart Guide*.

Writing programs for APPC conversations

Depending on which APPC conversation type you select, see:

- “Part 2. Writing programs for APPC mapped conversations” on page 21
- “Part 4. Writing programs for APPC basic conversations” on page 69.

Part 2. Writing programs for APPC mapped conversations

This is the first of four parts detailing the CICS APIs available for DTP programming.

- Part 2. Writing programs for APPC mapped conversations
- "Part 3. Writing programs for MRO conversations" on page 51
- "Part 4. Writing programs for APPC basic conversations" on page 69
- "Part 5. Writing programs for LUTYPE6.1 conversations" on page 103.

The different APIs are compared in "Part 1. Concepts and design considerations" on page 1.

Part 2 contains:

- "Chapter 3. APPC mapped conversation flow" on page 23.

  This advises you how to use the EXEC CICS API to write distributed transactions that use APPC mapped conversations.
- "Chapter 4. State transitions in APPC mapped conversations" on page 41.

  This discusses the state transitions that occur when transactions use APPC mapped conversations under the EXEC CICS API. The state transitions are presented in the form of state tables showing which commands can be issued while a conversation partner is in any given state. The tables also show how the conversation state changes as a result of issuing a command.

Chapter 3. APPC mapped conversation flow

This chapter 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 chapter contains the following topics:

- Starting the conversation
- “Transferring data on the conversation” on page 27
- “Communicating errors across a conversation” on page 32
- “Safeguarding data integrity” on page 32
- “Ending the conversation” on page 35
- “Checking the outcome of a DTP command” on page 37
- “Summary of CICS commands for APPC mapped conversations” on page 40.

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 conversation initiation failure. This section also contains program fragments illustrating the commands described below and the suggested response code checking.

Conversation initiation

The front-end transaction is responsible for acquiring a session, specifying the conversation characteristics and requesting the startup of the back-end transaction in the remote system.

Allocating a session to the conversation

Initially, there is no conversation, and therefore no conversation state. By issuing an 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), the conversation is in **allocated state** (state 1) and the session identifier (**convid**) in EIBRSRCE must be saved immediately.

The convid must be used in subsequent commands for this conversation. Figure 3 on page 24 shows an example of an ALLOCATE command.

Note: If the remote system is using VTAM persistent session support, you may need to code a timeout value on the ALLOCATE command. See “Using VTAM persistent session support” on page 18.

Using ATI to allocate a session

Front-end transactions are often initiated from terminals. But it is also possible to use the EXEC CICS START command to initiate a front-end transaction on an APPC session. When this is done, and the front-end transaction is successfully started, a conversation can continue as if an ALLOCATE command had been issued. The only difference is that, when ATI is used, the APPC session is the front-end transaction’s principal facility.

```
*    ...
 DATA DIVISION.
 WORKING-STORAGE SECTION.
*    ...
 01  FILLER.
     02  WS-CONVID       PIC X(4).
     02  WS-RESP         PIC S9(8) COMP.
     02  WS-STATE        PIC S9(8) COMP.
     02  WS-SYSID        PIC X(4) VALUE 'SYSB'.
     02  WS-PROC         PIC X(4) VALUE 'BBBB'.
     02  WS-LEN-PROCN    PIC S9(4) COMP VALUE +4.
     02  WS-SYNC-LVL     PIC S9(4) COMP VALUE +2.
*    ...
 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-CONVID)
                               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 3. Starting an APPC mapped conversation at sync level 2

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 CONNECT PROCESS. This command is used to attach the required back-end transaction. It should be noted that the results of the CONNECT PROCESS are placed in the send buffer and are not sent immediately to the partner system. Transmission occurs when the send buffer is flushed, either by sending more data than fits in the send buffer or by issuing a WAIT CONVID command.

A successful CONNECT PROCESS causes the conversation to switch to **send state** (state 2). The program fragment in Figure 3 shows an example of a CONNECT PROCESS command.

Note: For clarity, the EXEC CICS ALLOCATE and CONNECT PROCESS commands shown in Figure 3 identify the partner LU and transaction explicitly. To avoid doing this, you could use the PARTNER option of these commands. This specifies a set of definitions that include the names of the partner LU, the communication profile to be used on the session, and the partner transaction. Thus, in Figure 3, the PARTNER option could be used instead of SYSID on the EXEC CICS ALLOCATE command, and instead of

PROCNAME and PROCLENGTH on the EXEC CICS CONNECT PROCESS command. The advantage of using PARTNER is that it makes your DTP programs more maintainable: the details of each partner program can be held in a single definition. For details of the PARTNER resource, see the *CICS Resource Definition Guide*.

Initial data for the back-end transaction

While connecting the back-end transaction, the front-end transaction can send initial data to it. This kind of data, called **program initialization parameters** (PIPs), is placed in specially formatted structures and specified on the CONNECT PROCESS command. The PIPLIST (along with PIPLENGTH) option of the CONNECT PROCESS command is used to send PIPs to the back-end transaction.

To examine any PIPs received, the back-end transaction uses the EXTRACT PROCESS command.

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:

Figure 4. Format of PIP data.

PIP data consists of one or more subfields; each subfield contains

- *A halfword binary integer specifying the total length of the subfield in bytes*
- *A reserved halfword*
- *The PIP data itself*

The length includes the length field itself and the length of the reserved field; that is, if the PIP field is n bytes long, then the length field contains n + 4.

CICS inserts information into the reserved fields to make the PIP architecturally correct. The PIPLENGTH option must specify the total length of the PIP list and must be between 4 and 32763.

Back-end transaction initiation

The back-end transaction is initiated as a result of the front end transaction’s CONNECT PROCESS command. Initially, the back-end transaction should determine the convid. This is not strictly necessary because the session is the back-end transaction’s principal facility making the CONVID parameter optional 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 option improves program readability and problem determination.

Figure 5 on page 26 shows a fragment of a back-end transaction that obtains the conversation identifier. The example uses the ASSIGN command for this purpose; another way is to access the information in EIBTRMID.

The back-end transaction can also retrieve its transaction name by issuing the EXTRACT PROCESS command. In the example shown in Figure 5, CICS places the transaction name in WS-PROC and the length of the name in WS-LEN-PROCN. With the EXTRACT PROCESS, 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 ASSIGN and the 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 a 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.

```
*    ...
 DATA DIVISION.
 WORKING-STORAGE SECTION.
*    ...
 01  FILLER.
     02  WS-CONVID       PIC X(4).
     02  WS-STATE        PIC S9(7) COMP.
     02  WS-SYSID        PIC X(4) VALUE 'SYSB'.
     02  WS-PROC         PIC X(4) VALUE 'BBBB'.
     02  WS-LEN-PROCN    PIC S9(4) COMP VALUE +4.
     02  WS-SYNC-LVL     PIC S9(4) COMP VALUE +2.
*    ...
 01  FILLER.
     02  WS-RECORD       PIC X(100).
     02  WS-MAX-LEN      PIC S9(4) COMP VALUE +100.
     02  WS-RCVD-LEN     PIC S9(4) COMP 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 5. Startup of a back-end APPC mapped transaction at sync level 2

What happens if the back-end transaction fails to start

It is possible that the back-end transaction fails to start. However there is a transmission delay mechanism in APPC, which informs the front-end transaction of this fact when the session has been active long enough for responses from the back-end system to have been received. The front-end transaction is informed of this via a TERMERR condition in response to a DTP command. EIBERR, EIBFREE, and EIBERRCD are set (see Table 9 on page 37 for the possible values of EIBERRCD).

Before sending data, the front-end transaction should find out whether the back-end transaction has started successfully. One way of doing this is to issue a SEND CONFIRM command directly after the CONNECT PROCESS command. This causes the front-end transaction to suspend until the back-end transaction responds or the failure notification described above is received. SEND CONFIRM is discussed in “How to synchronize a conversation using CONFIRM commands” on page 33.

Transferring data on the 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 SEND command is valid only in **send state** (state 2). Because a successful simple SEND leaves the conversation in **send state** (state 2), it is possible to issue a number of successive sends. The data from the simple SEND command is initially stored in a local CICS buffer which is “flushed” either when this buffer is full or when the transaction requests transmission. The transaction can request transmission either by using a WAIT CONVID command or by using the WAIT option on the SEND command. The reason data transmission is deferred is to reduce the number of calls to the network. However, the application should use WAIT if the partner transaction requires the data to continue processing.

An example of a simple SEND command can be seen in Figure 6 on page 28.

```
*    ...
 DATA DIVISION.
 WORKING-STORAGE SECTION.
*    ...
 01  FILLER.
     02  WS-CONVID       PIC X(4).
     02  WS-STATE        PIC S9(7) COMP.
*    ...
 01  FILLER.
     02  WS-SEND-AREA    PIC X(70).
     02  WS-SEND-LEN     PIC S9(4) COMP VALUE +70.
*    ...
 01  FILLER.
     02  WS-RCVD-AREA    PIC X(100).
     02  WS-MAX-LEN      PIC S9(4) COMP VALUE +100.
     02  WS-RCVD-LEN     PIC S9(4) COMP 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 6. Transferring data on a conversation at sync level 2

Switching from sending to receiving data

The column for **send state** (state 2) in the state tables (see page 41) shows that there are several ways of switching from **send state** (state 2) to **receive state** (state 5).

One possibility is to use a RECEIVE command. The state tables show that CICS supplies the INVITE and WAIT when a SEND is followed immediately by a RECEIVE.

Another possibility is to use a SEND INVITE command. The state tables show that after SEND INVITE the conversation switches to **pendreceive state** (state 3). The column for state 3 shows that a WAIT CONVID command switches the conversation to **receive state** (state 5).

Still another possibility is to specify the INVITE and WAIT options on the SEND command. The state tables show that after SEND INVITE WAIT, the conversation switches to **receive state** (state 5).

An example of a SEND INVITE WAIT command can be seen in Figure 6 on page 28 . Figure 7 illustrates the response-testing sequence after a SEND INVITE WAIT with the STATE option. For more information on response testing, see "Checking the outcome of a DTP command" on page 37.

```
*    ...
 DATA DIVISION.
 WORKING-STORAGE SECTION.
*    ...
 01  FILLER.
     02  WS-RESP         PIC S9(7) COMP.
     02  WS-STATE        PIC S9(7) COMP.
*    ...
 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(ROLLBACK)
*                  ...  ROLLBACK received
             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 7. Checking the outcome of a SEND INVITE WAIT command

Receiving data from the partner transaction

The RECEIVE command is used to receive data from the connected partner. The rows in the state tables for the RECEIVE command show the EIB fields that should be tested after issuing a 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 8 on page 31. The conversation state can be meaningfully tested only after issuing a command with the STATE option or by using the 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 37. For information about testing the conversation state, see "Testing the conversation state" on page 50.

An example of a RECEIVE command with the STATE option can be seen in Figure 6 on page 28. Figure 8 on page 31 illustrates the response-testing and state-testing sequence.

Note: In the same way as it is possible to send the INVITE, LAST, and CONFIRM commands with data, it is also possible to receive them with data. It is also possible to receive a syncpoint request with data. However, ISSUE ERROR, ISSUE ABEND, and conversation failure are never received with data.

```
*    ...
 WORKING-STORAGE SECTION.
*    ...
 01  FILLER.
     02  WS-RESP         PIC S9(8) COMP.
     02  WS-STATE        PIC S9(8) COMP.
*    ...
 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(SYNCFREE)
*                 ...  Partner issued SYNCPOINT and LAST
             WHEN DFHVALUE(SYNCRECEIVE)
*                 ...  Partner issued SYNCPOINT
             WHEN DFHVALUE(SYNCSEND)
*                 ...  Partner issued SYNCPOINT and INVITE
             WHEN DFHVALUE(CONFFREE)
*                 ...  Partner issued CONFIRM and LAST
             WHEN DFHVALUE(CONFRECEIVE)
*                 ...  Partner issued CONFIRM
             WHEN DFHVALUE(CONFSEND)
*                 ...  Partner issued CONFIRM and INVITE
             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(ROLLBACK)
*                 ...  ROLLBACK received
             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 8. Checking the outcome of a RECEIVE command

The CONVERSE command

The CONVERSE command combines the functions SEND INVITE WAIT and RECEIVE. This command is useful when one transaction needs a response from the 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, ISSUE ABEND, causes the conversation to terminate abnormally and is described in "Emergency termination of a conversation" on page 36. The other two commands are described below.

Requesting INVITE from the partner transaction

If a transaction is receiving data on a conversation and wishes to send, it can use the ISSUE SIGNAL command to request that the partner transaction does a SEND INVITE. When the ISSUE SIGNAL request is received, EIBSIG=X'FF' and the SIGNAL condition is raised. It should be noted that on receipt of SIGNAL a transaction is **not** obliged to issue SEND INVITE.

Demanding INVITE from the partner transaction

If a transaction needs to send an immediate error notification to the partner transaction it can use the ISSUE ERROR command. This command is also one of the preferred negative responses to SEND CONFIRM. However it should **not** be used to reject ISSUE PREPARE, SYNCPOINT or SYNCPOINT ROLLBACK. When the ISSUE ERROR is received, EIBERR=X'FF' and the first two bytes of EIBERRCD are X'0889'. This error condition cannot be processed by HANDLE CONDITION (or RESP).

If an ISSUE ERROR command is used in **receive state** (state 5), all incoming data is purged until an INVITE, SYNCPOINT, or LAST command is received. If LAST 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 LAST is not received, the conversation is switched to **send state** (state 2). It is normal programming practice to communicate the reason for the ISSUE ERROR to the partner transaction. The CONVERSE command could be used to send an appropriate error message and receive a reply.

Because ISSUE ERROR is allowed in both **send state** (state 2) and **receive state** (state 5), it is possible for both communicating transactions to use ISSUE ERROR at the same time. When this occurs, only one of the ISSUE ERROR commands is effective. The other is purged with incoming data. However both ISSUE ERROR commands will appear to have completed successfully and the transaction whose ISSUE ERROR was purged will pick up EIBERR=X'FF' on a subsequent command.

Safeguarding data integrity

If it is important to safeguard data integrity across connected transactions, then the CICS synchronization commands shown in Table 5 are available.

Table 5. Synchronization commands for APPC mapped conversations

Conversation sync level	Commands
0	None
1	SEND CONFIRM ISSUE CONFIRMATION

Table 5. Synchronization commands for APPC mapped conversations (continued)

Conversation sync level	Commands
2	SEND CONFIRM ISSUE CONFIRMATION SYNCPOINT ISSUE PREPARE SYNCPOINT ROLLBACK SRRCMIT[2] SRRBACK[2]

The above commands are defined in the sections that follow.

How to synchronize a conversation using CONFIRM commands

A confirmation exchange affects a single specified conversation and involves only two commands:

1. The conversation that is in **send state** (state 2) issues a 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 ISSUE CONFIRMATION command. Alternatively, it can respond negatively by using the ISSUE ERROR or ISSUE ABEND commands.

The following sections describe these commands in more detail. The descriptions refer to the state tables for sync levels 1 and 2.

Requesting confirmation

The CONFIRM option of the SEND command flushes 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 SEND CONFIRM command. You can also specify either the INVITE or the LAST option.

The **send state** (state 2) column of the state table for APPC mapped conversations at sync level 1 on page 44 shows what happens for the possible combinations of the CONFIRM, INVITE, and LAST options. After a SEND CONFIRM command, without the INVITE or LAST options, the conversation remains in **send state** (state 2). If the INVITE option is used, the conversation switches to **receive state** (state 5). If the LAST option is used, the conversation switches to **free state** (state 12).

A similar effect to SEND LAST CONFIRM can by achieved by using the command sequence:

```
SEND LAST
SEND CONFIRM
```

Note from the state tables that the SEND LAST puts the conversation into **pendfree state** (state 4), so data cannot be sent with a 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 SEND CONFIRM **must** always be checked. See “Checking the response to SEND CONFIRM” on page 34.

2. SAA verbs for SYNCPOINT and SYNCPOINT ROLLBACK respectively.

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 6.

Table 6. Indications of a confirmation request

Command issued by partner transaction	Conversation state on receipt of request	EIBCONFon receipt of request	EIBRECVon receipt of request	EIBFREE on receipt of request
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 ISSUE CONFIRMATION command.
2. Reply negatively with an 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 ISSUE ABEND command. This makes the conversation unusable and a FREE command must be issued immediately.

Checking the response to SEND CONFIRM

After issuing SEND [INVITE|LAST] CONFIRM, it is important to test EIBERR to determine the partner's response. Table 7 on page 35 shows how the partner's response is indicated by EIB flags and the conversation states.

Table 7. Indications of responses to SEND CONFIRM

Command issued in reply by partner transaction	Conversation state on receipt of response	EIBERR on receipt of response	EIBFREE on receipt of response
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 ISSUE CONFIRMATION.

If the partner replies ISSUE ERROR, this is indicated by EIBERR=X'FF' and the first two bytes of EIBERRCD = X'0889'. When the partner replies ISSUE ERROR in response to SEND LAST CONFIRM, the LAST option is ignored and the conversation is **not** terminated. The conversation state is switched to **receive state** (state 5).

If the partner replies ISSUE ABEND, your transaction will be abended AZCH. In addition, EIBERR and EIBFREE are set, and the first two bytes of EIBERRCD=X'0864'. The conversation is switched to **free state**.

How to synchronize conversations using SYNCPOINT commands

Data synchronization (the SYNCPOINT and SYNCPOINT ROLLBACK commands) affects all connected conversations at sync level 2. The use of these commands in DTP is described in “Part 6. Syncpointing a distributed process” on page 117.

Ending the conversation

The following sections describe the different ways a conversation can end, either unexpectedly or under transaction control. To end a conversation, one transaction issues a request for termination and the other receives this request. Once this has happened the conversation is unusable and **both** transactions must issue a FREE command to release the session.

Normal termination of a conversation

The SEND LAST command is used to terminate a conversation. It should be used in conjunction with either the WAIT or CONFIRM options, the SYNCPOINT command, or the WAIT CONVID command (depending on the conversation sync level). This is described in Table 8.

Table 8. Command sequences for ending a conversation

Sync level	Command sequence
0	SEND LAST WAIT FREE
1	SEND LAST CONFIRM FREE
2	SEND LAST[3] SYNCPOINT FREE

From the state tables it can be seen that it is possible to end a conversation by issuing the FREE command, provided the conversation is in **send state** (state 2). This will generate an implicit SEND LAST WAIT command before the FREE is executed and is therefore not recommended for conversations using sync levels 1 and 2.

Note: A distributed transaction should not end a conversation by issuing an EXEC CICS RETURN command, but instead follow the sequence of commands shown in Table 8 on page 35. The issue of an EXEC CICS RETURN could lead to one or both transactions ending abnormally.

Emergency termination of a conversation

The ISSUE ABEND command provides a means of abnormally ending the conversation. It is valid for all levels of synchronization, but should be avoided at sync level 2, because its use at the wrong time can lead to a loss of data integrity.

ISSUE ABEND 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 flushed before the ISSUE ABEND command is transmitted.

The transaction that issues the ISSUE ABEND command is not itself abended. It must, however, issue a FREE command for the conversation unless it is designed to terminate immediately.

If an 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 ISSUE ABEND is received, CICS abends the transaction with abend code AZCH, sets on EIBERR(=X'FF'),EIBFREE(=X'FF'), and places X'0864' in the first two bytes of EIBERRCD.

Unexpected termination of a conversation

If a partner system fails, or a session goes out of service in the middle of a DTP conversation, the conversation is terminated abnormally and the TERMERR condition is raised on the next command that accesses the conversation. In addition, EIBERR and EIBFREE are set on (X'FF') and EIBERRCD contains a value representing the reason for the error, as follows:

X'08640001' - partner system with persistent session support has failed and restarted
X'1008600B' - session has failed due to a protocol error
X'A0000100' - temporary session failure
X'A0010100' - RTIMOUT time-out value was exceeded.

3. It is important that the SEND LAST command for sync level 2 is **not** accompanied by WAIT or CONFIRM because either of these options will cause the conversation to end before the subsequent syncpoint has propagated to the partner transaction. This may mean that protected resources of one transaction could be committed while those in the partner transaction could be backed out. The resulting state errors may also lead to the session being unbound.

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 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.

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 9 shows how these EIB fields interact.

Table 9. Interaction between some EIB fields—all DTP commands

EIB- ERR	EIB- FREE	EIB- SYNRB	EIBERRCD	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

Table 9. Interaction between some EIB fields—all DTP commands (continued)

EIB- ERR	EIB- FREE	EIB- SYNRB	EIBERRCD	Description
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 10 on page 39 shows how some of these EIB fields interact for RECEIVE and CONVERSE commands.

Table 10. 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 4. State transitions in APPC mapped conversations" on page 41.

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 7 on page 29 and Figure 8 on page 31.

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 11 shows the CICS commands used in APPC mapped conversations.

Table 11. Summary of CICS commands used in mapped conversations

Use to ...	Sync levels	CICS command	Page
Acquire a session.	0,1,2	ALLOCATE	23
Initiate a conversation.	0,1,2	CONNECT PROCESS	24
Access session-related information.	0,1,2	EXTRACT PROCESS	25
Send data and control information to the conversation partner.	0,1,2	SEND	27
Receive data from the conversation partner.	0,1,2	RECEIVE	29
Send and receive data on the conversation.	0,1,2	CONVERSE	31
Transmit any deferred data or control indicators.	0,1,2	WAIT CONVID	27
Reply positively to SEND CONFIRM.	1,2	ISSUE CONFIRMATION	34
Prepare a conversation partner for syncpointing.	2	ISSUE PREPARE	120
Inform the conversation partner of a program-detected error.	0,1,2	ISSUE ERROR	32
Signal an unusual condition to the conversation partner, usually against the flow of data.	0,1,2	ISSUE SIGNAL	32
Inform the conversation partner that the conversation should be abandoned.	0,1,2	ISSUE ABEND	36
Free the session.	0,1,2	FREE	35
Inform all conversation partners of readiness to commit changes to recoverable resources.	2	SYNCPOINT	119
Inform conversation partners of the need to back out changes to recoverable resources.	2	SYNCPOINT ROLLBACK	120

For programming information about CICS commands, see the *CICS Application Programming Reference* manual.

Chapter 4. State transitions in APPC mapped conversations

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 chapter contains the following topics:
- “The state tables for APPC mapped conversations”
- “Testing the conversation state” on page 50.

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.
×	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.

Table 12. APPC mapped conversations at sync level 0, part 1

Command issued	EIB flag returned[4]	ALLO-CATED[11]	SEND	PEND-RECEIVE	PEND-FREE	RECEIVE	CONF-RECEIVE
		State 1	State 2	State 3	State 4	State 5	State 6
CONNECT PROCESS	EIBERR + EIBFREE	12	Ab	Ab	Ab	Ab	N/A
CONNECT PROCESS [13]	×	2	Ab	Ab	Ab	Ab	N/A
EXTRACT PROCESS[5]	×	=	=	=	=	=	N/A
EXTRACT ATTRIBUTES	×	=	=	=	=	=	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	×	Ab	5	Ab	Ab	Ab	N/A
SEND INVITE	×	Ab	3	Ab	Ab	Ab	N/A
SEND LAST WAIT	×	Ab	12	Ab	Ab	Ab	N/A
SEND LAST	×	Ab	4	Ab	Ab	Ab	N/A
SEND WAIT	×	Ab	=	Ab	Ab	Ab	N/A
SEND	×	Ab	=	Ab	Ab	Ab	N/A
RECEIVE	EIBERR + EIBFREE	Ab	12[7]	12[10]	Ab	12	N/A
RECEIVE	EIBERR	Ab	5[7]	5[10]	Ab	=	N/A
RECEIVE	EIBFREE	Ab	12[7]	12[10]	Ab	12	N/A
RECEIVE	EIBRECV	Ab	5[7]	5[10]	Ab	=	N/A
RECEIVE NOTRUNCATE[6]	EIBCOMPL[6]	Ab	5[7]	5[10]	Ab	=	N/A
RECEIVE	×	Ab	=[7]	2[10]	Ab	2	N/A
CONVERSE[8]	EIB flags and states as for RECEIVE						
ISSUE ERROR	EIBFREE	Ab	12	12	Ab	12	N/A
ISSUE ERROR	×	Ab	=	2	Ab	2	N/A
ISSUE ABEND	×	Ab	12	12	12	12	N/A
ISSUE SIGNAL[12]	×	Ab	=	=	Ab	=	N/A
WAIT CONVID	×	Ab	=	5	12	Ab	N/A
FREE	×	End	End [9]	Ab	End	Ab	N/A

Note: See page 49 for footnotes.

Table 13. APPC mapped conversations at sync level 0, part 2

CONF-SEND	CONF-FREE	SYNC-RECEIVE	SYNC-SEND	SYNC-FREE	FREE	ROLL-BACK	
State 7	**State 8**	**State 9**	**State 10**	**State 11**	**State 12**	**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

Table 14. APPC mapped conversations at sync level 1, part 1

Command issued	EIB flag returned[4]	ALLO-CATED[11] 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[13]	×	2	Ab	Ab	Ab	Ab	Ab
EXTRACT PROCESS[5]	×	Ab	=	=	=	=	=
EXTRACT ATTRIBUTES	×	=	=	=	=	=	=
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	×	Ab	5	Ab	Ab	Ab	Ab
SEND INVITE CONFIRM	×	Ab	5	Ab	Ab	Ab	Ab
SEND INVITE	×	Ab	3	Ab	Ab	Ab	Ab
SEND LAST WAIT	×	Ab	12	Ab	Ab	Ab	Ab
SEND LAST CONFIRM	×	Ab	12	Ab	Ab	Ab	Ab
SEND LAST	×	Ab	4	Ab	Ab	Ab	Ab
SEND WAIT	×	Ab	=	Ab	Ab	Ab	Ab
SEND CONFIRM	×	Ab	=	5	12[14]	Ab	Ab
SEND	×	Ab	=	Ab	Ab	Ab	Ab
RECEIVE	EIBERR + EIBFREE	Ab	12[7]	12[10]	Ab	12	Ab
RECEIVE	EIBERR	Ab	5[7]	5[10]	Ab	=	Ab
RECEIVE	EIBCONF + EIBFREE	Ab	8[7]	8[10]	Ab	8	Ab
RECEIVE	EIBCONF + EIBRECV	Ab	6[7]	6[10]	Ab	6	Ab
RECEIVE	EIBCONF	Ab	7[7]	7[10]	Ab	7	Ab
RECEIVE	EIBFREE	Ab	12[7]	12[10]	Ab	12	Ab
RECEIVE	EIBRECV	Ab	5[7]	5[10]	Ab	=	Ab
RECEIVE NOTRUNCATE[6]	EIBCOMPL[6]	Ab	5[7]	5[10]	Ab	=	Ab
RECEIVE	×	Ab	=[7]	2[10]	Ab	2	Ab
CONVERSE[8]	EIB flags and states as for RECEIVE						
ISSUE CONFIRMATION	×	Ab	Ab	Ab	Ab	Ab	5
ISSUE ERROR	EIBFREE	Ab	12	12	Ab	12	12
ISSUE ERROR	×	Ab	=	2	Ab	2	2
ISSUE ABEND	×	Ab	12	12	12	12	12
ISSUE SIGNAL[12]	×	Ab	=	=	Ab	=	=
WAIT CONVID	×	Ab	=	5	12	Ab	Ab
FREE	×	End	End[9]	Ab	End	Ab	Ab

Note: See page 49 for footnotes.

Table 15. APPC mapped conversations at sync level 1, part 2

CONF-SEND	CONF-FREE	SYNC-RECEIVE	SYNC-SEND	SYNC-FREE	FREE	ROLL-BACK	
State 7	State 8	State 9	State 10	State 11	State 12	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 16. APPC mapped conversations at sync level 2, part 1

Command issued	EIB flag returned[4]	ALLO-CATED[11]	SEND	PEND-RECEIVE	PEND-FREE	RECEIVE	CONF-RECEIVE
		State 1	**State 2**	**State 3**	**State 4**	**State 5**	**State 6**
CONNECT PROCESS	EIBERR + EIBFREE	12	Ab	Ab	Ab	Ab	Ab
CONNECT PROCESS[13]	×	2	Ab	Ab	Ab	Ab	Ab
EXTRACT PROCESS[5]	×	=	=	=	=	=	=
EXTRACT ATTRIBUTES	×	=	=	=	=	=	=
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	×	Ab	5	Ab	Ab	Ab	Ab
SEND INVITE CONFIRM	×	Ab	5	Ab	Ab	Ab	Ab
SEND INVITE	×	Ab	3	Ab	Ab	Ab	Ab
SEND LAST WAIT[15]	×	Ab	12	Ab	Ab	Ab	Ab
SEND LAST CONFIRM[15]	×	Ab	12	Ab	Ab	Ab	Ab
SEND LAST	×	Ab	4	Ab	Ab	Ab	Ab
SEND WAIT	×	Ab	=	Ab	Ab	Ab	Ab
SEND CONFIRM	×	Ab	=	5[14]	12[14]	Ab	Ab
SEND	×	Ab	=	Ab	Ab	Ab	Ab
RECEIVE	EIBERR + EIBSYNRB	Ab	13[7]	13[10]	Ab	13	Ab
RECEIVE	EIBERR + EIBFREE	Ab	12[7]	12[10]	Ab	12	Ab
RECEIVE	EIBERR	Ab	5[7]	5[10]	Ab	=	Ab
RECEIVE	EIBSYNC + EIBFREE	Ab	11[7]	11[10]	Ab	11	Ab
RECEIVE	EIBSYNC + EIBRECV	Ab	9[7]	9[10]	Ab	9	Ab
RECEIVE	EIBSYNC	Ab	10[7]	10[10]	Ab	10	Ab
RECEIVE	EIBCONF + EIBFREE	Ab	8[7]	8[10]	Ab	8	Ab
RECEIVE	EIBCONF + EIBRECV	Ab	6[7]	6[10]	Ab	6	Ab
RECEIVE	EIBCONF	Ab	7[7]	7[10]	Ab	7	Ab
RECEIVE	EIBFREE	Ab	12[7]	12[10]	Ab	12	Ab
RECEIVE	EIBRECV	Ab	5[7]	5[10]	Ab	=	Ab
RECEIVE NOTRUNCATE[6]	EIBCOMPL[6]	Ab	5[7]	5[10]	Ab	=	Ab
RECEIVE	×	Ab	=[7]	2[10]	Ab	2	Ab
CONVERSE[8]	EIB flags and states as for RECEIVE						

Note: See page 49 for footnotes.

Table 17. APPC mapped conversations at sync level 2, part 2

CONF-SEND	CONF-FREE	SYNC-RECEIVE	SYNC-SEND	SYNC-FREE	FREE	ROLL-BACK	
State 7	State 8	State 9	State 10	State 11	State 12	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
Ab	Ab	Ab	Ab	Ab	Ab	Ab	After confirm flow detected
Ab	Ab	Ab	Ab	Ab	Ab	Ab	After error flow detected
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......

Table 18. APPC mapped conversations at sync level 2, part 3

Command issued	EIB flag returned[4]	ALLO-CATED[11]	SEND	PEND-RECEIVE	PEND-FREE	RECEIVE	CONF-RECEIVE
		State 1	**State 2**	**State 3**	**State 4**	**State 5**	**State 6**
ISSUE CONFIRMATION	×	Ab	Ab	Ab	Ab	Ab	5
ISSUE ERROR	EIBFREE	Ab	12	12	Ab	12	12
ISSUE ERROR	×	Ab	=	2	Ab	2	2
ISSUE ABEND	×	Ab	12	12	12	12	12
ISSUE SIGNAL[12]	×	Ab	=	=	Ab	=	=
ISSUE PREPARE	EIBERR + EIBSYNRB	Ab[19]	13	13	13	Ab[19]	Ab[19]
ISSUE PREPARE	EIBERR + EIBFREE	Ab[19]	12	12	12	Ab[19]	Ab[19]
ISSUE PREPARE	EIBERR	Ab[19]	5	5	5	Ab[19]	Ab[19]
ISSUE PREPARE	×	Ab[19]	10[21]	9[21]	11[21]	Ab[19]	Ab[19]
SYNCPOINT[17]	EIBRLDBK	=	2 or 5[18]	2 or 5[18]	2 or 5[18]	Ab[20]	Ab[20]
SYNCPOINT[17]	×	=	=	5	12	Ab[20]	Ab[20]
SYNCPOINT ROLLBACK[17]	×	=	2 or 5[18]	2 or 5[18]	2 or 5[18]	2 or 5[18]	2 or 5[18]
WAIT CONVID	×	Ab	=	5	12	Ab	Ab
FREE	×	End	End[9]	Ab	End	Ab	Ab

4. 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 occur 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.
5. You can issue the EXTRACT PROCESS command from the back-end transaction only.
6. 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 small to contain all the data received, CICS truncates the data and sets the LENGERR condition.
7. Equivalent to SEND INVITE WAIT followed by RECEIVE.
8. Equivalent to SEND INVITE WAIT [FROM] followed by RECEIVE.
9. Equivalent to SEND LAST WAIT followed by FREE.
10. Equivalent to WAIT followed by RECEIVE.
11. 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**.
12. ISSUE SIGNAL sets the partner's EIBSIG flag.
13. The back-end transaction starts in **receive state** after the front-end transaction has issued CONNECT PROCESS.
14. No data may be included with SEND CONFIRM.
15. Although CICS allows you to terminate a sync level-2 conversation using the SEND LAST WAIT or SEND LAST CONFIRM commands, doing this deviates from the APPC architecture and should be avoided. See "CICS deviations from the APPC architecture" on page 161.

Table 19. APPC mapped conversations at sync level 2, part 4

CONF-SEND	CONF-FREE	SYNC-RECEIVE	SYNC-SEND	SYNC-FREE	FREE	ROLL-BACK	
State 7	State 8	State 9	State 10	State 11	State 12	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
=	=	=[16]	=[16]	=[16]	Ab	Ab	Immediately
Ab	Ab	Ab	Ab	Ab	Ab	Ab	After response from partner
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 response from partner
Ab[20]	Ab[20]	2 or 5[18]	2 or 5[18]	2 or 5[18]	=	Ab[20]	After response from partner
Ab[20]	Ab[20]	5	2	12	=	Ab[20]	After response from partner
2 or 5[18]	2 or 5[18]	2 or 5[18]	2 or 5[18]	2 or 5[18]	=	2 or 5[18]	After rollback across UOW
Ab	Ab	Ab	Ab	Ab	Ab	Ab	Immediately
Ab	Ab	Ab	Ab	Ab	End	Ab	Immediately

16. Where APPC transaction routing is taking place, the ISSUE SIGNAL command is invalid in this state.
17. 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, including MRO conversations.
18. The state of each conversation after rollback depends on several factors:
    - The system you are communicating with. Some earlier versions of CICS handle rollback differently from CICS Transaction Server for z/OS, Version 2 Release 2.
    - 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 Transaction Server for z/OS, Version 2 Release 2 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 has issued a SEND LAST WAIT or FREE commands.

    After a syncpoint or rollback, it is advisable to determine the conversation state before issuing any further commands against the conversation.
19. This results, not in an ATCV abend, but in an INVREQ return code.
20. This causes an ASP2 abend, not an ATCV.
21. 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 20 shows how the cvda codes relate to the conversation state. The table also shows the symbolic names defined for these cvda values.

Table 20. 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
Syncreceive	9	DFHVALUE(SYNCRECEIVE)	92
Syncsend	10	DFHVALUE(SYNCSEND)	93
Syncfree	11	DFHVALUE(SYNCFREE)	91
Free	12	DFHVALUE(FREE)	85
Rollback	13	DFHVALUE(ROLLBACK)	89

Part 3. Writing programs for MRO conversations

This is the second of four parts detailing the CICS APIs available for DTP programming.

- "Part 2. Writing programs for APPC mapped conversations" on page 21
- Part 3. Writing programs for MRO conversations
- "Part 4. Writing programs for APPC basic conversations" on page 69
- "Part 5. Writing programs for LUTYPE6.1 conversations" on page 103.

The different APIs are compared in "Part 1. Concepts and design considerations" on page 1.

Part 3 contains:

- "Chapter 5. MRO conversation flow" on page 53.

  This advises you how to use the EXEC CICS API to write distributed transactions that use MRO conversations.
- "Chapter 6. State transitions in MRO conversations" on page 65.

  This discusses the state transitions that occur when transactions use MRO conversations under the EXEC CICS API. The state transitions are presented in the form of a state table that shows which commands can be issued while the conversation is in any given state. The table shows how the conversation state changes as a result of issuing a command.

Chapter 5. MRO conversation flow

This chapter introduces some of the MRO DTP commands. 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 chapter contains the following topics:

- Starting the conversation
- “Transferring data on the conversation” on page 56
- “Safeguarding data integrity” on page 60
- “Ending the conversation” on page 60
- “Checking the outcome of a DTP command” on page 61
- “Summary of commands for MRO conversations” on page 63.

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. The third subsection discusses the possibility of conversation initiation failure. This section also contains program fragments illustrating the commands described and the suggested response code checks.

Conversation initiation

The front-end transaction is responsible for acquiring a session, specifying the conversation characteristics and requesting the startup of the back-end transaction in the partner system.

Allocating a session to the conversation

Initially, there is no conversation, and therefore no conversation state. By issuing an 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 successfully allocated, DFHRESP(NORMAL), the conversation is in **allocated state** (state 1) and the session identifier (**convid**) from EIBRSRCE must be saved immediately.

The convid must be used in subsequent commands for this conversation. Figure 9 on page 55 shows a program fragment containing an example of the ALLOCATE command. You will notice that the PROFILE option has been omitted from the command.

If the PROFILE option is specified for an MRO link, CICS ignores it at execution time. So none of the facilities selected through use of a profile (for example, RTIMEOUT and JOURNALING) are available. The front-end transaction has no control over its session processing options when an MRO session is being used.

A back-end transaction with an MRO session as its principal facility will be sent the INBFMH parameter by CICS, regardless of the what the front-end transaction specifies on the PROFILE option of the ALLOCATE command.

Using ATI to allocate a session

Front-end transactions are often initiated from terminals. But it is also possible to use the EXEC CICS START command to initiate a front-end transaction on an MRO session. When the front-end transaction is successfully started in this way, a conversation can continue as if an ALLOCATE command had been issued. The only difference is that an automatically-initiated front-end transaction has the MRO session as its *principal facility*.

Connecting the partner transaction

When a session has been acquired, the next step is to cause the partner transaction to be initiated. The state table shows that, in **allocated state** (state 1), one of the commands available is SEND. Using this command, the back-end transaction's identifier can be specified in the first four bytes of the data which, when transferred to the partner system, will be used to attach the required back-end transaction. The send buffer containing the transaction identifier together with any other data, will be flushed immediately and the front-end transaction will wait until a response is received from the back end. Figure 9 on page 55 shows an example in which a transaction identifier is sent.

Alternatively, when a session has been acquired, the front-end transaction can build and send an attach header with the first transmission of data. The attach header can be built using the BUILD ATTACH command.

When using the BUILD ATTACH command, an eight-character name must be given to the built attach header which can then be used in the ATTACHID option of the first SEND (or CONVERSE) command. The back-end transaction identifier should also be specified.

```
*    ...
 DATA DIVISION.
 WORKING-STORAGE SECTION.
*    ...
 01  FILLER.
     02  WS-CONVID       PIC X(4).
     02  WS-RESP         PIC S9(8) COMP.
     02  WS-STATE        PIC S9(8) COMP.
     02  WS-SYSID        PIC X(4) VALUE 'SYSB'.
     02  WS-PROC         PIC X(4) VALUE 'BBBB'.
     02  WS-LEN-PROCN    PIC S9(5) COMP VALUE +4.
*    ...
 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 EIBRCODE.
     END-IF.
*    ...
     EXEC CICS SEND CONVID(WS-CONV) RESP(WS-RESP) STATE(WS-STATE)

                    FROM(WS-PROC) LENGTH(WS-LEN-PROCN)
     END-EXEC.
     IF WS-RESP = DFHRESP(NORMAL)
     THEN
*       ...  No errors, conversation started.
     ELSE
*       ...  Conversation not started.  Examine EIBRCODE.
     END-IF.
```

Figure 9. Starting an MRO conversation

Back-end transaction initiation

The back-end transaction is initiated either by an attach header received from the partner system or by a transaction identifier included in the incoming data, and is started with the session as its principal facility. Initially, the back-end transaction should determine the convid from EIBTRMID. This is not strictly necessary because the session is the back-end transaction's principal facility making the CONVID parameter optional for DTP commands on this conversation. However, the convid is very useful for audit trails. Also, if the back-end transaction is involved in more than one conversation, then always specifying the convid will improve program readability and problem determination. Figure 10 on page 56 shows a back-end transaction that does obtain the convid.

When the back-end transaction receives data, the presence of an attach header is indicated by either EIBATT or RESP(INBFMH). One of these is normally set after the back-end transaction issues its first RECEIVE command. The EXTRACT ATTACH command can be used to access session-related information from the attach header (for example, the back-end transaction identifier) if required, but it is not mandatory.

```
*    ...
 DATA DIVISION.
 WORKING-STORAGE SECTION.
*    ...
 01  FILLER.
     02  WS-CONVID       PIC X(4).
     02  WS-STATE        PIC S9(7) COMP.
*    ...
 01  FILLER.
     02  WS-RECORD       PIC X(100).
     02  WS-MAX-LEN      PIC S9(5) COMP VALUE +100.
     02  WS-RCVD-LEN     PIC S9(5) COMP VALUE +0.
*    ...
 PROCEDURE DIVISION.
*    ...
     EXEC CICS ASSIGN FACILITY(WS-CONVID) 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 10. Startup of a back-end MRO transaction

What happens if the back-end transaction fails to start

It is possible that the back-end transaction may fail to start up. This will result in the front-end transaction abending. Message DFHIR3783 contains the reason for the error.

Transferring data on the conversation

This section discusses how to pass data between the front-end 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 includes an example program fragment, which illustrates the commands described and the suggested response code checking.

Sending data to the partner transaction

The SEND command is used to send data to the connected partner. This command is valid in **allocated state** (state 1) or **send state** (state 2). Because a successful simple SEND completes in **send state** (state 2), it is possible to issue a number of successive sends.

An example of a simple SEND command can be seen in Figure 11 on page 57.

```
*    ...
 DATA DIVISION.
 WORKING-STORAGE SECTION.
*    ...
 01  FILLER.
     02  WS-CONVID       PIC X(4).
     02  WS-RESP         PIC S9(8) COMP.
     02  WS-STATE        PIC S9(8) COMP.
*    ...
 01  FILLER.
     02  WS-SEND-AREA    PIC X(70).
     02  WS-SEND-LEN     PIC S9(5) COMP VALUE +70.
*    ...
 01  FILLER.
     02  WS-RCVD-AREA    PIC X(100).
     02  WS-MAX-LEN      PIC S9(5) COMP VALUE +100.
     02  WS-RCVD-LEN     PIC S9(5) COMP VALUE +0.
*    ...
 PROCEDURE DIVISION.
*    ...
     EXEC CICS SEND CONVID(WS-CONVID) RESP(WS-RESP)
                    STATE(WS-STATE)
                    FROM(WS-SEND-AREA) LENGTH (WS-SEND-LEN)
     END-EXEC.
*    ... Check outcome of SEND.
*    ...
*
     EXEC CICS SEND INVITE CONVID(WS-CONVID)
                    RESP(WS-RESP) STATE(WS-STATE)
     END-EXEC.
*    ...
*    Receive data from the partner transaction.
*
     EXEC CICS RECEIVE CONVID(WS-CONVID)
                       RESP(WS-RESP) STATE(WS-STATE)
                       INTO(WS-RCVD-AREA) MAXLENGTH(WS-MAX-LEN)
                       NOTRUNCATE LENGTH(WS-RCVD-LEN)
     END-EXEC.
*
*    ...  Check outcome of RECEIVE.
*    ...
```

Figure 11. Transferring data on an MRO conversation

Switching from sending to receiving data

The column for **send state** (state 2) in the state table in “Chapter 6. State transitions in MRO conversations” on page 65 shows that there is only one way of switching from **send state** (state 2) to **receive state** (state 5). That is to use a SEND INVITE command with or without the WAIT option. The state table shows that after both SEND INVITE and SEND INVITE WAIT, the conversation switches the current state to **receive state** (state 5).

An example of a SEND INVITE command can be seen in Figure 11.

```
*    ...
 DATA DIVISION.
 WORKING-STORAGE SECTION.
*    ...
 01  FILLER.
     02  WS-RESP         PIC S9(8) COMP.
     02  WS-STATE        PIC S9(8) COMP.
*    ...
 PROCEDURE DIVISION.
*    ...
* Check return code from SEND INVITE
     IF WS-RESP = DFHRESP(NORMAL)
     THEN
*       ...  Request successful, check state
        IF WS-STATE = DFHVALUE(RECEIVE)
        THEN
*          ...  SEND OK, continue processing
        ELSE
*          ...  Logic error, should never happen
        END-IF
     ELSE
*       ...  Examine EIBRCODE for source of error
     END-IF.
*    ...
```

Figure 12. Checking the outcome of a SEND INVITE command

Receiving data from the partner transaction

The RECEIVE command is used to receive data from the connected partner. The rows in the state tables for the RECEIVE command show the EIB fields that should be tested after issuing a RECEIVE command. As well as showing which field should be tested, the state table also shows the order in which the tests should be made. Instead of testing some of the EIB fields, you can test the resulting conversation state; this is shown in Figure 13 on page 59. Note that you should always test the value returned by the RESP option.

```
*    ...
 DATA DIVISION.
 WORKING-STORAGE SECTION.
*    ...
 01  FILLER.
     02  WS-RESP         PIC S9(8) COMP.
     02  WS-STATE        PIC S9(8) COMP.
*    ...
 PROCEDURE DIVISION.
*    ...
* Check return code from RECEIVE
     IF WS-RESP = DFHRESP(NORMAL)
     THEN
*       ...  Request successful, check state
        EVALUATE WS-STATE
          WHEN DFHVALUE(ROLLBACK)
*              ...  Partner issued SYNCPOINT ROLLBACK
          WHEN DFHVALUE(SYNCFREE)
*              ...  Partner issued SYNCPOINT and LAST
          WHEN DFHVALUE(SYNCRECEIVE)
*              ...  Partner issued SYNCPOINT
          WHEN DFHVALUE(FREE)
*              ...  Partner issued LAST
          WHEN DFHVALUE(SEND)
*              ...  Partner issued INVITE
          WHEN DFHVALUE(RECEIVE)
*              ...  Processing for receipt of data
*                   (including EIBCOMPL for incomplete data)
          WHEN OTHER
*              ...  Logic error, should never happen
        END-EVALUATE.
     ELSE
*       ...  Examine EIBRCODE for source of error
     END-IF.
*    ...
```

Figure 13. Checking the outcome of a RECEIVE command

Note: In the same way as it is possible to send the INVITE and LAST indicators with data, it is also possible to receive them with data. Syncpoint requests may also be received with data. However, indications of conversation failure are never received with data.

The CONVERSE command

The CONVERSE command combines the functions SEND INVITE and RECEIVE. This command is useful when one transaction needs a response from the partner transaction to continue processing.

Safeguarding data integrity

If it is important to safeguard data integrity across connected transactions, then the following synchronization commands are available:

SYNCPOINT SYNCPOINT ROLLBACK SRRCMIT (SAA verb for SYNCPOINT) SRRBACK (SAA verb for SYNCPOINT ROLLBACK).

The use of these commands in DTP is described in “Part 6. Syncpointing a distributed process” on page 117.

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. Once this has happened the conversation is unusable and **both** transactions must issue a FREE command to release the session.

Ending a conversation normally

The SEND LAST command is used to terminate a conversation. It should be used in conjunction with either the WAIT option or the SYNCPOINT command, and followed by the FREE command. However, SEND LAST WAIT causes the conversation to end before any subsequent syncpoint can be propagated to the partner transaction. This may mean that the protected resources in one system could be committed whilst those in the other system could be backed out.

From the state table it can be seen that it is possible to end a conversation by issuing the FREE command provided the conversation is in **send state** (state 2). This generates an implicit SEND LAST WAIT command before the FREE is executed and therefore is not recommended.

Note: A distributed transaction should not end a conversation by issuing an EXEC CICS RETURN command, but instead follow the sequence of commands described above. The issue of an EXEC CICS RETURN could lead to one or both transactions ending abnormally.

Unexpected termination of a conversation

If a partner systems fails, or a session goes out of service in the middle of a DTP conversation, the transaction is terminated abnormally.

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 is the same as for other EXEC CICS commands in that conditions are raised and may be handled using HANDLE CONDITION or RESP. EIBRCODE will also contain an error code.

If the request has not failed, it is 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 MRO DTP commands. (See the *CICS Application Programming Reference* manual for programming information on the contents and format of EIB fields.)

EIBFREE
when set to X'FF' indicates that the partner transaction has ended the conversation. It should be tested in conjunction with EIBSYNC to determine exactly how to end the conversation.

EIBSYNC
when set to X'FF' indicates the partner transaction has requested a syncpoint.

EIBSYNRB
when set to X'FF' indicates the partner transaction has issued a SYNCPOINT ROLLBACK command.

Table 21 shows how these EIB fields interact.

Table 21. Interaction of some EIB fields

EIB- FREE	EIB- SYNRB	EIB- SYNC	Description
X'00'	X'FF'	X'00'	The partner transaction or system has issued SYNCPOINT ROLLBACK.
X'FF'	X'00'	X'00'	The partner transaction or system has issued SEND LAST followed by a FREE command.
X'FF'	X'00'	X'FF'	The partner transaction or system has issued SEND LAST followed by SYNCPOINT. The local program should reply with a SYNCPOINT command followed by a FREE command.
X'00'	X'00'	X'FF'	The partner transaction or system has issued a SYNCPOINT.

In addition the following EIB fields are relevant only to the RECEIVE and CONVERSE commands:

EIBATT
when set to X'FF' indicates that the data received contained an attach header. The attach header is not passed to the application; however, EIBATT indicates that an EXTRACT ATTACH command is appropriate.

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.

EIBFMH
: when set to X'FF' indicates that the data passed to the application contains a concatenated Function Management Header (FMH). This happens only when the partner CICS transaction builds an FMH in the data and the FMH option on the SEND command is specified.

EIBRECV
: when set to X'00' indicates the partner transaction used the INVITE or LAST option on its last SEND command. When set on (X'FF'), EIBRECV indicates that another RECEIVE is required.

After the EIB fields have been analyzed, it is possible to test the conversation state to determine which DTP commands may be issued next. See "Chapter 6. State transitions in MRO conversations" on page 65.

Note: CICS ignores the profile you specify on the PROFILE option of the ALLOCATE for an MRO link and instead uses the default profile. This enables FMHs to be sent and received and EIBATT or EIBFMH to be set appropriately. The default profile DFHCICSA, used for the session allocated by the front-end transaction, has INBFMH (ALL) specified. The default principal facility profile DFHCICST used for the back-end transaction does not have INBFMH (ALL) specified.

Checking EIB fields and the conversation state

Most of the information supplied by the EIB indicator fields can be obtained from the conversation state. However, there are some EIB fields that you cannot ignore. For example, when the conversation remains in **receive state** (state 5) after a RECEIVE command has been issued, only EIBFMH indicates that the partner transaction has sent an FMH.

Note that the state table provided in "Chapter 6. State transitions in MRO conversations" on page 65 contains not only states and commands issued, but also relevant EIB fields settings. The order in which the EIB fields are shown provides a sensible sequence for checking them in an application.

Summary of commands for MRO conversations

Table 22 shows the commands used in MRO conversations.

Table 22. Summary of CICS commands used in MRO conversations

Use to ...	Command	Page
Acquire a session.	ALLOCATE	53
Build an attach header.	BUILD ATTACH	54
Access session-related information.	EXTRACT ATTACH	55
Send data and control information to the conversation partner.	SEND	56
Receive data from the conversation partner.	RECEIVE	58
Send and receive data on the conversation.	CONVERSE	60
Inform all conversation partners of readiness to commit recoverable resources.	SYNCPOINT	119
Inform conversation partners of the need to back out changes to recoverable resources.	SYNCPOINT ROLLBACK	120
Free the session.	FREE	60

For programming information about CICS commands, see the *CICS Application Programming Reference* manual.

Chapter 6. State transitions in MRO conversations

This chapter shows the state transitions that occur when transactions engage in MRO conversations. The state transitions are presented in the form of a state table. The state table shows which commands a transaction can issue while the conversation is in any given state. It also shows how the conversation state changes as a result of any command.

The state table for MRO conversations

The state table provides the following information for writing a DTP program. Firstly, it shows which commands can be issued from each conversation state. Secondly, it shows the results of issuing a command in terms of state transactions and EIB fields.

How to use the state table

The commands you can issue, coupled with the EIB flags that can be set after execution, are shown down the left side of the table. These commands correspond to the rows of the table. 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. The order in which EIB flags are shown with a command is the order in which you should test the EIB flags in your program.

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.
×	The EIB flag is any one that has not been covered in earlier rows, or it is irrelevant.
Ab	The command is not valid in this state. Issuing a command in a state in which it is not valid usually causes an AZI1 abend.
=	Remains in current state.
End	End of conversation.

Table 23. MRO conversation states, part 1

Command issued	EIB flag returned[4]	ALLO-CATED[11]	SEND	PEND-RECEIVE	PEND-FREE	RECEIVE	CONF-RECEIVE
		State 1	State 2	State 3	State 4	State 5	State 6
BUILD ATTACH	×	=	=	N/A	=	Ab	N/A
EXTRACT ATTACH	×	=	=	N/A	=	=	N/A
EXTRACT ATTRIBUTES	×	=	=	N/A	=	=	N/A
SEND INVITE WAIT[23]	×	5	5	N/A	Ab	Ab	N/A
SEND INVITE	×	5	5	N/A	Ab	Ab	N/A
SEND LAST WAIT[23]	×	12	12	N/A	Ab	Ab	N/A
SEND LAST	×	4	4	N/A	Ab	Ab	N/A
SEND	×	2	=	N/A	Ab	Ab	N/A
RECEIVE	EIBSYNC + EIBFREE + EIBCOMPL	Ab	Ab	N/A	Ab	11	N/A
RECEIVE	EIBSYNC + EIBRECV + EIBCOMPL	Ab	Ab	N/A	Ab	9	N/A
RECEIVE	EIBSYNRB + EIBCOMPL	Ab	Ab	N/A	Ab	13	N/A
RECEIVE	EIBFREE	Ab	Ab	N/A	Ab	12	N/A
RECEIVE	EIBRECV	Ab	Ab	N/A	Ab	=	N/A
RECEIVE NOTRUNCATE[24]	EIBCOMPL[24]	Ab	Ab	N/A	Ab	=	N/A
RECEIVE	×	Ab	Ab	N/A	Ab	2	N/A
CONVERSE[25]	EIB flags and states as for RECEIVE but allowed in send state						
SYNCPOINT[29]	EIBRLDBK	=	2 or 5[28]	N/A	2 or 5[28]	Ab[22]	N/A
SYNCPOINT[29]	×	=	=	N/A	12	Ab[22]	N/A
SYNCPOINT ROLLBACK[29]	×	=	2 or 5[28]	N/A	2 or 5[28]	2 or 5[28]	N/A
FREE	×	End[26]	End[26]	N/A	End	Ab	N/A

22. This causes an ASP1 abend, not AZI1.

23. The option WAIT on the SEND command does not flush data on MRO conversations. But it may affect the move to the next state.

24. 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 small to contain all the data received, CICS truncates the data and sets the LENGERR condition.

25. Equivalent to:

```
SEND INVITE [FROM]
RECEIVE
```

26. Equivalent to:

```
SEND LAST WAIT
FREE
```

Table 24. MRO conversation states, part 2

CONF-SEND	CONF-FREE	SYNC-RECEIVE	SYNC-SEND	SYNC-FREE	FREE	ROLL-BACK	
State 7	**State 8**	**State 9**	**State 10**	**State 11**	**State 12**	**State 13**	**Command returns**
N/A	N/A	=	N/A	=	=	=	Immediately
N/A	N/A	=	N/A	=	=	=	Immediately
N/A	N/A	=	N/A	=	=	=	Immediately
N/A	N/A	Ab	N/A	Ab	Ab	Ab	After data and CD flows
N/A	N/A	Ab	N/A	Ab	Ab	Ab	After data and CD flows
N/A	N/A	Ab	N/A	Ab	Ab	Ab	After data and EB flows
N/A	N/A	Ab	N/A	Ab	Ab	Ab	After data flows
N/A	N/A	Ab	N/A	Ab	Ab	Ab	After data flows
N/A	N/A	Ab	N/A	Ab	Ab	Ab	After sync flow detected
N/A	N/A	Ab	N/A	Ab	Ab	Ab	After sync flow detected
N/A	N/A	Ab	N/A	Ab	Ab	Ab	After rollback flow detected
N/A	N/A	Ab	N/A	Ab	Ab	Ab	After EB detected
N/A	N/A	Ab	N/A	Ab	Ab	Ab	When data available
N/A	N/A	Ab	N/A	Ab	Ab	Ab	When data available
N/A	N/A	Ab	N/A	Ab	Ab	Ab	When data available
States as for RECEIVE							When data available
N/A	N/A	2 or 5[28]	N/A	2 or 5[28]	=	Ab	After response from partner
N/A	N/A	5	N/A	12	=	Ab	After response from partner
N/A	N/A	2 or 5[28]	N/A	2 or 5[28]	=	2 or 5[28]	After rollback across UOW
N/A	N/A	Ab	N/A	Ab	End	Ab	Immediately

27. Before a session is allocated, there is no conversation, and therefore no conversation state. The ALLOCATE command does not appear in the table. 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**.

    You select the partner transaction program by issuing a SEND command or a CONVERSE command. You have the choice of identifying the transaction program either in the first four bytes of the user data or in the attach function management header built by the BUILD ATTACH command.

    The back-end transaction starts in **receive state**.

28. The state of each conversation after rollback depends on several factors:
    - The system you are communicating with. Some earlier versions of CICS handle rollback differently from CICS Transaction Server for z/OS, Version 2 Release 2.
    - 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 Transaction Server for z/OS, Version 2 Release 2 follows the architecture.

    Always use the EXTRACT ATTRIBUTES STATE command or the STATE option on the EXEC CICS commands to determine the conversation state.

29. The commands SYNCPOINT and SYNCPOINT ROLLBACK do not relate to any particular conversation, but are propagated on all the conversations that are currently active for the task, including APPC conversations.

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'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 an application to inquire on the current conversation state. 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 25 shows how the cvda codes relate to the conversation state. It also shows the symbolic names defined for the cvda values.

Table 25. 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
Pendfree	4	DFHVALUE(PENDFREE)	86
Receive	5	DFHVALUE(RECEIVE)	88
Syncreceive	9	DFHVALUE(SYNCRECEIVE)	92
Syncfree	11	DFHVALUE(SYNCFREE)	91
Free	12	DFHVALUE(FREE)	85
Rollback	13	DFHVALUE(ROLLBACK)	89

Part 4. Writing programs for APPC basic conversations

This is the third of four parts detailing the CICS APIs available for DTP programming.

- “Part 2. Writing programs for APPC mapped conversations” on page 21
- “Part 3. Writing programs for MRO conversations” on page 51
- Part 4. Writing programs for APPC basic conversations
- “Part 5. Writing programs for LUTYPE6.1 conversations” on page 103.

The different APIs are compared in “Part 1. Concepts and design considerations” on page 1.

Part 4 contains:

- “Chapter 7. APPC basic conversation flow” on page 71.

  This describes how to write APPC basic conversations using the EXEC CICS GDS interface. To use this interface, the application must insert the data to be sent into GDS (generalized data stream) records and extract it from records received. This part describes the format of GDS records and a possible strategy for building them.

  Note that CICS applications that use the APPC basic interface can be written only in assembler language or C.
- “Chapter 8. State transitions in APPC basic conversations” on page 93.

  This discusses the state transitions that occur when transactions use APPC basic conversations under the EXEC CICS GDS API. The state transitions are presented in the form of state tables showing which commands can be issued while a conversation partner is in any given state. The tables also show how the conversation state changes as a result of issuing a command.

For further information about the APPC architecture, see the *Peer Protocols* manual, SC30-3269 and the *LU6.2 Reference: Verb Descriptions* manual, GC30-3084. For information about the mapping between APPC verbs and CICS commands, see “Appendix A. CICS mapping to the APPC architecture” on page 147.

Chapter 7. APPC basic conversation flow

This chapter introduces some of the GDS commands. It introduces each command in the context of a typical conversation flow and ends with a general discussion of how to test the outcome of a GDS command. Although the examples are given in assembler, it is also possible to write C programs for APPC basic conversations.

The chapter contains the following topics:

- Starting the conversation
- “Sending data to the partner transaction” on page 75
- “Receiving data from the partner transaction” on page 77
- “Communicating errors across a conversation” on page 80
- “Safeguarding data integrity” on page 81
- “Ending the conversation” on page 83
- “Checking the outcome of GDS commands” on page 85
- “Summary of commands for APPC basic conversations” on page 91.

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 conversation initiation failure. This section also contains program fragments illustrating the commands described and the suggested response code checking.

Conversation initiation

The front-end transaction is responsible for acquiring a session, specifying the conversation characteristics, and requesting the startup of the back-end transaction in the partner system.

Allocating a session to the conversation

Initially, there is no conversation, and therefore no conversation state. By issuing a GDS ALLOCATE command, the front-end transaction acquires a session to start a new conversation.

RETCODE should be checked to ensure that a session has really been allocated. If successfully allocated (RETCODE = X'00'), the conversation is in **allocated state** (state 1) and the session identifier (**convid**) is placed in the data area specified on the CONVID parameter.

The convid must be used in subsequent commands for this conversation. Figure 14 on page 72 shows an example of a GDS ALLOCATE command.

Note: If the remote system is using VTAM persistent session support, you may need to code a timeout value on the GDS ALLOCATE command. See “Using VTAM persistent session support” on page 18.

Using ATI to allocate a session

Front-end transactions are often initiated from terminals. But it is also possible to use the EXEC CICS START command to initiate a front-end transaction on an APPC session. When this is done, and the front-end transaction is successfully started, a conversation can continue as if a GDS ALLOCATE command had been issued. The only difference is that, when ATI is used, the APPC session is the front-end transaction’s principal facility.


```
*         ...
          EXEC CICS GDS ALLOCATE SYSID(WSYSID) CONVID(WCONVID)         *

                                 STATE(WSTATE) RETCODE(WRETC)
*
*         Check outcome of GDS ALLOCATE
*
          NC    WRETC,WRETC
          BNZ   ALLOCERR          No session allocated, check RETCODE
*         ...
          EXEC CICS GDS CONNECT PROCESS CONVID(WCONVID) STATE(WSTATE)  *
                                        PROCNAME(WPROC)                *
                                        PROCLENGTH(WLENPROC)           *
                                        SYNCLEVEL(WSYNCLVL)            *

                                        CONVDATA(WCDB) RETCODE(WRETC)
          NC    WRETC,WRETC
          BNZ   CONNERR           Request failed, analyze RETCODE
*         ...                     No errors, conversation started.
          NC    CDBERR,CDBERR
          BNZ   SESSERR           Session failed, examine RETCODE.
*         ...                     Start sending data.
*         ...
WSTATE    DS    F
WRETC     DS    XL6
WCDB      DS    0CL24
          COPY  DFHCDBLK
WCONVID   DS    CL4
WSYSID    DC    CL4'SYSB'
WPROC     DC    CL4'BBBB'
WLENPROC  DC    F'4'
WSYNCLVL  DC    F'2'
*         ...
```

Figure 14. Starting an APPC basic conversation at sync level 2

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 GDS CONNECT PROCESS. This command is used to attach the required back-end transaction. It should be noted that the results of the GDS CONNECT PROCESS are placed in the send buffer and are not sent immediately to the partner system. Transmission occurs when the send buffer is flushed, either by sending more data than fits in the send buffer or by issuing a GDS WAIT command.

A successful GDS CONNECT PROCESS causes the conversation state to switch to **send state** (state 2). Figure 14 is a program fragment showing an example of a GDS CONNECT PROCESS.

Note: For clarity, the EXEC CICS GDS ALLOCATE and GDS CONNECT PROCESS commands shown in Figure 14 identify the partner LU and transaction explicitly. To avoid doing this, you could use the PARTNER option of these commands. This specifies a set of definitions that include the names of the partner LU, the communication profile to be used on the session, and the partner transaction. Thus, in Figure 14, the PARTNER option could be used instead of SYSID on the EXEC CICS GDS ALLOCATE command, and instead of PROCNAME and PROCLENGTH on the EXEC CICS GDS CONNECT PROCESS command. The advantage of using PARTNER is that it makes your DTP programs more maintainable: the details of each partner program can be held in a single definition. For details of the PARTNER resource, see the *CICS Resource Definition Guide*.

Initial data for the back-end transaction

While connecting the back-end transaction, the front-end transaction can send initial data to it. This kind of data, called **program initialization parameters** (PIPs), is placed in specially formatted structures and specified on the GDS CONNECT PROCESS command. The PIPLIST (along with PIPLENGTH) option of the GDS CONNECT PROCESS command is used to send PIPs to the back-end transaction.

To examine any PIPs received, the back-end transaction uses the GDS EXTRACT PROCESS command.

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:

L1	rr	PIP1	L2	rr	PIP2		Ln	rr	PIPn

Figure 15. Format of PIP data.

PIP data consists of one or more subfields; each subfield contains

- *A halfword binary integer specifying the total length of the subfield in bytes*
- *A reserved halfword*
- *The PIP data itself*

The length includes the length field itself and the length of the reserved field; that is, if the PIP field is n bytes long, then the length field contains n + 4.

CICS inserts information in the reserved fields so that the PIP is architecturally correct. The PIPLENGTH option must specify the total length of the PIP list and must be between 4 and 32763.

Back-end transaction initiation

A back-end transaction is initiated as a result of the front end's GDS CONNECT PROCESS command. Initially the back-end transaction should determine the convid. Figure 16 on page 74 shows a fragment of a back-end transaction that uses the EXEC CICS GDS ASSIGN command to obtain the convid. The back-end transaction can also obtain the transaction identifier and sync level used to start the conversation. The GDS EXTRACT PROCESS command is used to obtain this information.

The back-end transaction starts in **receive state** (state 5). So, after obtaining the convid, the back-end transaction can issue a GDS RECEIVE command.

```
*         ...
          EXEC CICS GDS ASSIGN PRINCONVID(WCONVID) RETCODE(WRETC)
*
*         ...
*
          EXEC CICS GDS EXTRACT PROCESS CONVID(WCONVID)               *
                                  PROCNAME(WPROC) RETCODE(WRETC) *
                                  PROCLENGTH(WLENPROC)           *

                                  SYNCLEVEL(WSYNCLVL)
*         ...
* Receive first data from front-end transaction.
*         ...
*
WSTATE    DS    F
WRETC     DS    XL6
WCDB      DS    0CL24
          COPY  DFHCDBLK
WCONVID   DS    CL4
WPROC     DS    CL4
WLENPROC  DS    F
WSYNCLVL  DS    F
*         ...
```

Figure 16. Startup of a back-end transaction

What happens if the back-end transaction fails to start up

It is possible that the back-end transaction fails to start up. However, because of the transmission delay mechanism in APPC, the front-end transaction is not informed of this fact until the conversation has been active long enough for responses from the back-end system to be received. The front-end transaction is informed of this via CDBERR and CDBFREE. In addition, CDBERRCD is set as shown in Table 26.

Table 26. Some indications of back-end failure

CDBERRCD value	Reason
10086032	The PIP data sent with the GDS CONNECT PROCESS was incorrectly specified.
10086034	The partner system does not support basic conversations.
080F6051	The partner transaction failed security check.
10086041	The partner transaction does not support the sync level requested on the GDS CONNECT PROCESS.
10086021	The partner system does not recognize the requested transaction identifier.
084C0000	The partner system cannot start the partner transaction.
084B6031	The partner system is temporarily unable to start the partner transaction.

Before sending data, the front-end transaction should find out whether the back end transaction has started successfully. One way of doing this is to issue a GDS SEND CONFIRM command directly after the GDS CONNECT PROCESS. This causes the front-end transaction to be suspended until the back end transaction has responded or the back-end system has sent the failure notification described above.

Sending data to the partner transaction

To send data on an APPC basic conversation, an application must format the data into **generalized data stream** (GDS) records. A GDS record contains a 16-bit (2-byte) header followed by the application data. The 16 bits of the header consist of the following fields:

Concatenation bit
: This is the high-order bit of the first byte of the header. An application program can use it to group records together logically. This bit does not affect the way CICS processes the records.

LL This is the rest of the header (15 bits). It specifies the overall length of the data (including the length of the header).

Figure 17 shows the format of GDS records.
Up to 32 765 bytes of application data can be accommodated in one GDS record.

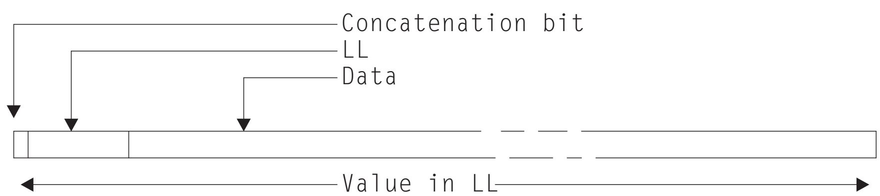

Figure 17. Format of GDS records

Data formatted into GDS records can be transmitted by the GDS SEND command. This command is valid only in **send state** (state 2).

Because a simple GDS SEND keeps the conversation in **send state** (state 2), you can issue a number of successive sends. You need not issue a GDS SEND for every record to be sent; you can send partial or multiple records at a time. However, make sure that the last logical record is complete when you use the INVITE, LAST, or CONFIRM options, and before you issue a syncpoint request.

Figure 18 is an example of the use of GDS SEND commands.

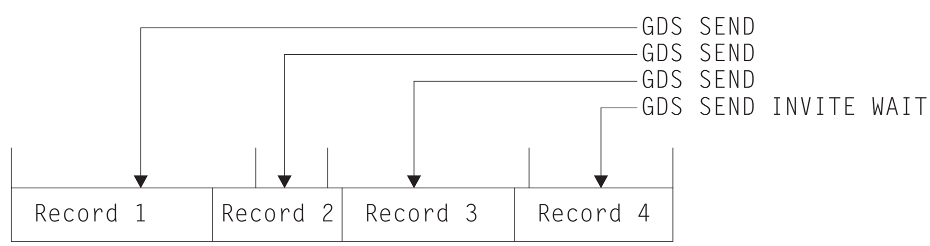

Figure 18. An example of the use of GDS SEND commands.

The data to be sent consists of four logical records:

1. *A GDS SEND command is used to transmit the whole of record 1, and the first portion of record 2.*
2. *A GDS SEND command is used to transmit the second portion of record 2.*
3. *A GDS SEND command is used to transmit the remaining portion of record 2, the whole of record 3, and the first portion of record 4.*
4. *A GDS SEND INVITE WAIT command is used to transmit the remaining portion of record 4.*

This flexibility also allows you to use separate GDS SEND commands for the GDS header and the application data—a useful technique to avoid shifting data into storage contiguous with its GDS header. The program fragment in Figure 19 uses this technique.

```
*         ...
          LA    R5,L'SENDHDR+LEN'SENDDATA  Compute LL value
          STH   R5,SENDHDR           Place length in LL
          LA    R5,L'SENDHDR         Length of GDS header
          ST    R5,SENDLEN            into send length field
          EXEC CICS GDS SEND FROM(SENDHDR) FLENGTH(SENDLEN)           *
                             CONVID(WCONVID) RETCODE(WRETC)           *

                             STATE(WSTATE) CONVDATA(WCDB)
*
*         ...                        Check outcome of the SEND
*         ...
          LA    R5,L'SENDDATA        Length of application data
          ST    R5,SENDLEN            into send length field
          EXEC CICS GDS SEND FROM(SENDDATA) FLENGTH(SENDLEN)          *
                             CONVID(WCONVID) RETCODE(WRETC)           *

                             STATE(WSTATE) CONVDATA(WCDB)
*
*         ...                        Check outcome of the SEND
*         ...
          EXEC CICS GDS SEND INVITE WAIT                              *
                             CONVID(WCONVID) RETCODE(WRETC)           *

                             STATE(WSTATE) CONVDATA(WCDB)
*
*         ...                        Check outcome of SEND INVITE WAIT

*         ...
*
WSTATE    DS    F
WRETC     DS    XL6
WCDB      DS    0CL24
          COPY  DFHCDBLK
WCONVID   DS    CL4
SENDDATA  DS    CL100
SENDLEN   DS    F
SENDHDR   DS    H
*         ...
```

Figure 19. Sending data on an APPC basic conversation

The records from a simple GDS SEND command are initially stored in a local CICS buffer which is “flushed” either when this buffer is full or when the transaction requests transmission. The transaction can request transmission either by using a GDS WAIT command or by using the WAIT option on the GDS SEND command. The reason transmission is deferred is to reduce the number of calls to the network. However, the application should use GDS WAIT if the partner transaction requires the data to continue processing.

Switching from sending to receiving data

To switch from sending to receiving records, use a GDS SEND INVITE command with the WAIT or CONFIRM option. This switches the conversation from **send state** (state 2) to **receive state** (state 5). An example of a GDS SEND INVITE WAIT command can be seen in Figure 19. Figure 25 on page 91 illustrates the response-testing sequence.

For further information on the CONFIRM option, see "How to synchronize conversations using CONFIRM commands" on page 81.

Receiving data from the partner transaction

The GDS RECEIVE command is used to receive data from the connected partner transaction. The rows in the state tables for the GDS RECEIVE command show the CONVDATA fields that should be tested after issuing a GDS RECEIVE command. As well as showing which fields should be tested, the state tables also show the order in which the tests should be made. As an alternative to testing some of the CONVDATA fields it is possible to test the resulting conversation state. This is shown in Figure 24 on page 90. Note that both RETCODE and CDBERR should always be tested.

The amount of data received is determined by:

- How much the conversation partner sent
- The value supplied on the MAXFLENGTH option
- Whether the LLID or BUFFER option is used.

The first factor is obvious: the application cannot receive more than is sent. The value of MAXFLENGTH is an upper limit; CICS never returns more bytes than this value specifies. The LLID and BUFFER options enable the application to specify how CICS is to treat the data. This is described in Receiving data by the record and "Receiving data by the buffer" on page 79.

In the same way as it is possible to send GDS records with the INVITE, LAST, or CONFIRM option, it is also possible to receive them together. Syncpoint requests can also be received with GDS records. However, GDS ISSUE ERROR, GDS ISSUE ABEND, and indications of conversation failure are received by themselves —never with GDS records.

An example of a GDS RECEIVE command can be seen in Figure 20 on page 78. Figure 24 on page 90 illustrates the response testing sequence.

```
*         ...
RECVLOOP  DS    0H
          LA    R5,L'RECVHDR         Length of GDS header
          ST    R5,RECVMAX            as maximum receive length
* Receive GDS header from partner transaction
          EXEC CICS GDS RECEIVE INTO(RECVHDR) MAXFLENGTH(RECVMAX)      *
                                LLID FLENGTH(RECVLEN)                  *
                                CONVID(WCONVID) RETCODE(WRETC)         *

                                STATE(WSTATE) CONVDATA(WCDB)
*
*         ...                        Check outcome of the GDS RECEIVE
*         ...
          LA    R5,L'RECVAREA        Length of application buffer
          ST    R5,RECVMAX            as maximum receive length
* Receive application data from partner transaction
          EXEC CICS GDS RECEIVE INTO(RECVAREA) MAXFLENGTH(RECVMAX)     *
                                LLID FLENGTH(RECVLEN)                  *
                                CONVID(WCONVID) RETCODE(WRETC)         *

                                STATE(WSTATE) CONVDATA(WCDB)
*         ...
*         ...                        Check outcome of the GDS RECEIVE
*         ...                        (including CDBCOMPL).
          B     RECVLOOP             Loop while in receive state
*         ...
*
WSTATE    DS    F
WRETC     DS    XL6
WCDB      DS    0CL24
          COPY  DFHCDBLK
WCONVID   DS    CL4
RECVAREA  DS    CL100
RECVMAX   DS    F
RECVLEN   DS    F
RECVHDR   DS    H
*         ...
```

Figure 20. Receiving data on an APPC basic conversation

Receiving data by the record

If you specify the LLID option on a GDS RECEIVE command, the data is considered as a series of GDS records. On each GDS RECEIVE request, data is received from not more than one record. If the record is longer than the value specified in the MAXFLENGTH option, two or more RECEIVE commands are required to recover the whole record. CDBCOMPL is set on when the end of a GDS record has been received. Consider the example shown in Figure 21 on page 79.

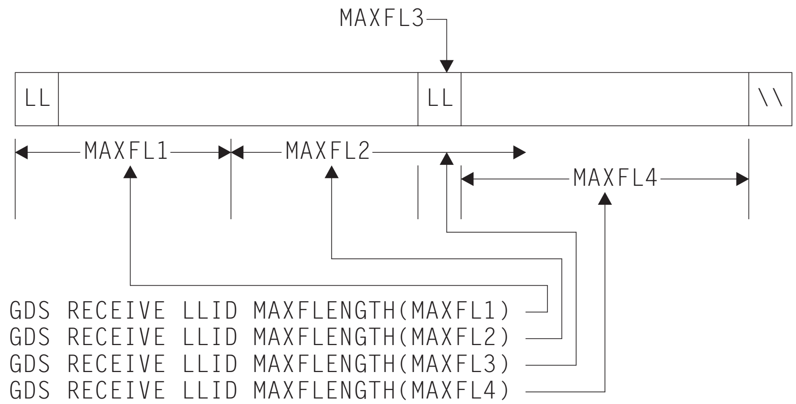

Figure 21. An example of the effect of the LLID option.

The data to be received consists of two logical records:

1. *A GDS RECEIVE LLID command specifying MAXFLENGTH(maxfl1) is issued. This returns the first portion of the first record. CDBCOMPL is set to X'00', indicating that a complete record has not been received.*
2. *A GDS RECEIVE LLID comand specifying MAXFLENGTH(maxfl2) is issued; because maxfl2 exceeds the length of the remaining data contained in the first logical record, the remaining data from the record is returned. CDBCOMPL is set to X'FF', indicating that a complete record has been received.*
3. *A GDS RECEIVE LLID command specifying MAXFLENGTH(maxfl3) is issued, where has a value of 2. The LL field from the second logical record is returned. CDBCOMPL is set to X'00', indicating that a complete record has not been received.*
4. *A GDS RECEIVE LLID command specifying MAXFLENGTH(maxfl4) is issued, wheremaxfl4 is the length of the remaining data in the second logical record. CDBCOMPL is set to X'FF', indicating that a complete record has been received.*

The first RECEIVE command receives the front portion of the first record. The length received is restricted by the MAXFLENGTH value (MAXFL1). The second RECEIVE command receives the rest of the first logical record. Even though the MAXFLENGTH value (MAXFL2) allows more data to be received, this cannot be done without breaking the LL boundary rule. The third RECEIVE command is for two bytes of data (the LL field). The fourth RECEIVE command receives the rest of the second record.

The application can tell if a complete record has been received, because CDBCOMPL is set (X'FF'). So, in the example given above, CDBCOMPL is set on after the second and fourth RECEIVE commands. CDBCOMPL is set off (X'00') after the first and third RECEIVE commands.

Receiving data by the buffer

Unlike the LLID option, the BUFFER option does not respect GDS record boundaries. If the MAXFLENGTH value allows, bytes will be received for more than one record. A GDS RECEIVE command with the BUFFER option recovers the length of data specified in the MAXFLENGTH option, ignoring GDS record boundaries. CICS does not return control to the application program until this length of data has been received or the partner transaction sends the INVITE or LAST option.

Figure 22 on page 80 shows the effect of the BUFFER option on the same four RECEIVE commands discussed in "Receiving data by the record" on page 78.

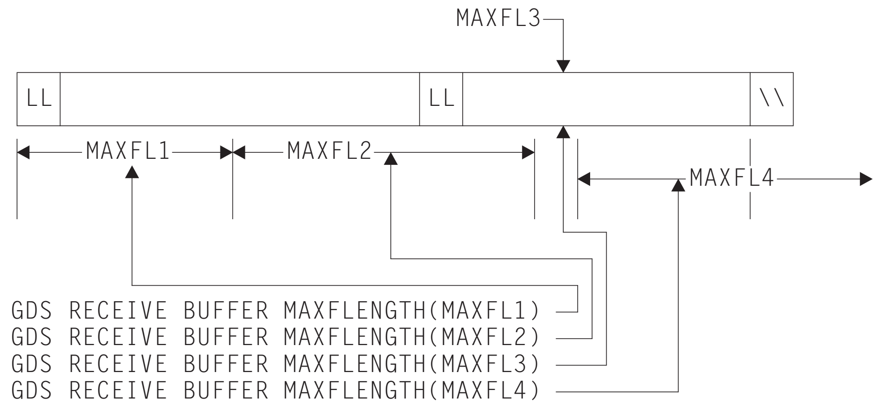

Figure 22. An example of the effect of the BUFFER option.

The data to be received consists of two logical records:

1. *A GDS RECEIVE BUFFER command specifying MAXFLENGTH(maxfl1) is issued. This returns the first portion of the first record.*
2. *A GDS RECEIVE BUFFER comand specifying MAXFLENGTH(maxfl2) is issued; because maxfl2 exceeds the length of the remaining data contained in the first logical record, the remaining data from the first record, and the first part of the second record (including the LL field), are returned.*
3. *A GDS RECEIVE LLID command specifying MAXFLENGTH(maxfl3) is issued, where has a value of 2. Two further bytes from the second logical record are returned.*
4. *A GDS RECEIVE LLID command specifying MAXFLENGTH(maxfl4) is issued, wheremaxfl4exceeds the length of the remaining data in the second logical record. The application waits until the partner transaction sends either enough data to satisfy the RECEIVE request, or the INVITE or LAST option.*

Communicating errors across a conversation

The APPC basic 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, GDS ISSUE ABEND, causes the conversation to terminate abnormally and is described in "Emergency termination of a conversation" on page 84. The other two commands are described in the following section.

Requesting INVITE from the partner transaction

If a transaction is receiving data on a conversation and wishes to send, it can use the GDS ISSUE SIGNAL command to request that the partner transaction does a GDS SEND INVITE. When the GDS ISSUE SIGNAL request is received, CDBSIG is set (X'FF'). Note that on receipt of a signal, a transaction is *not* obliged to issue GDS SEND INVITE.

Demanding INVITE from the partner transaction

If a transaction wishes to send an immediate error notification to the partner transaction it can use the GDS ISSUE ERROR command. This command is also one of the preferred negative responses to GDS SEND CONFIRM. However it should **not** be used to reject GDS ISSUE PREPARE, SYNCPOINT or SYNCPOINT ROLLBACK. When the GDS ISSUE ERROR is received, CDBERR is set (X'FF') and the first two bytes of CDBERRCD are X'0889'.

If a GDS ISSUE ERROR command is used in **receive state** (state 5), all incoming data is purged until an INVITE, SYNCPOINT or LAST is received. If LAST is received, no error indication is sent to the partner transaction, CDBFREE is set (X'FF') and the conversation is switched to **free state** (state 12).

If LAST is not received, the conversation is switched to **send state** (state 2). It is normal to communicate the reason for the error to the partner transaction. The GDS SEND INVITE WAIT command could be used to send an appropriate error message and then a GDS RECEIVE could be used to receive a reply.

Because GDS ISSUE ERROR is allowed in both **send state** (state 2) and **receive state** (state 5), it is possible for both communicating transactions to use GDS ISSUE ERROR at the same time. When this happens, only one of the GDS ISSUE ERROR commands is effective. The other is purged with incoming data. However, both commands will appear to have completed successfully and the transaction whose GDS ISSUE ERROR was purged will pick up CDBERR (=X'FF') on a subsequent command.

Safeguarding data integrity

If it is important to safeguard data integrity across connected transactions, then the CICS synchronization commands shown in Table 27 are available.

Table 27. Synchronization commands for APPC basic applications

Conversation sync level	Commands
0	None
1	GDS SEND CONFIRM GDS ISSUE CONFIRMATION
2	GDS SEND CONFIRM GDS ISSUE CONFIRMATION SYNCPOINT GDS ISSUE PREPARE SYNCPOINT ROLLBACK SRRCMIT[30] SRRBACK[30]

These commands are defined in the following sections.

How to synchronize conversations using CONFIRM commands

A confirmation exchange affects a single, specified, conversation and involves only two commands:

1. The transaction that is in **send state** (state 2) issues a GDS SEND CONFIRM command causing a request for confirmation to be sent to the partner transaction. The transaction is suspended awaiting a response.
2. The partner transaction receives a request for confirmation. It can then respond positively by issuing a GDS ISSUE CONFIRMATION command. Alternatively, it can respond negatively by using the GDS ISSUE ERROR or GDS ISSUE ABEND commands.

30. SAA verbs for SYNCPOINT and SYNCPOINT ROLLBACK respectively.

The following sections describe these commands in more detail. The descriptions refer to the state tables for sync levels 1 and 2.

Requesting confirmation

The CONFIRM option on the GDS SEND command flushes the conversation send buffer; that is, it causes a real transmission to occur.

Data can be sent with the GDS SEND CONFIRM command. Either the INVITE or the LAST option can also be specified.

The **send state** (state 2) column of the state table for APPC basic conversations at sync level 1 on page 96 shows what happens for the possible combinations of the CONFIRM, INVITE, and LAST options. After a GDS SEND CONFIRM command, without the INVITE or LAST options, the conversation remains in **send state** (state 2). If the INVITE option is used, the conversation switches to **receive state** (state 5). If the LAST option is used, the conversation switches to **free state** (state 12).

A similar effect to GDS SEND LAST CONFIRM can by achieved by using the command sequence:

```
GDS SEND LAST
GDS SEND CONFIRM
```

Note from the state tables that the GDS SEND LAST puts the conversation into **pendfree state** (state 4), so data cannot be sent with a GDS 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. Whichever is used, the response from GDS SEND CONFIRM *must* always be checked. (See “Checking the response to GDS SEND CONFIRM” on page 83.)

Receiving and replying to a confirmation request

On receipt of a confirmation request, the CONVDATA and conversation state will be set depending on the request issued by the partner transaction. These together with the contents of the CDBCONF, CDBRECV, and CDBFREE fields are shown in Table 28.

Table 28. How confirmation requests affect the state and flags

Command issued by partner transaction	Conversation state on receipt of request	CDB-CONF on receipt of request	CDB-RECV on receipt of request	CDB-FREE on receipt of request
GDS SEND CONFIRM	confreceive (state 6)	X'FF'	X'FF'	X'00'
GDS SEND INVITE CONFIRM	confsend (state 7)	X'FF'	X'00'	X'00'
GDS SEND LAST CONFIRM	conffree (state 8)	X'FF'	X'00'	X'FF'

There are three ways of replying:

1. Reply positively with a GDS ISSUE CONFIRMATION command.
2. Reply negatively with a GDS 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 a GDS ISSUE ABEND command. This makes the conversation unusable and a GDS FREE command must be issued immediately.

Checking the response to GDS SEND CONFIRM

After issuing GDS SEND [INVITE|LAST] CONFIRM, it is important to test CDBERR to determine the partner transaction's response. Table 29 shows the response received when the partner transaction issues different commands.

Table 29. Indicators of the partner transaction's response

Command issued in reply by partner transaction	Conversation state	CDBERR	CDBFREE
GDS ISSUE CONFIRMATION	Dependent on original GDS SEND [INVITE|LAST] CONFIRM request	X'00'	X'00'
GDS ISSUE ERROR	Receive (state 5)	X'FF'	X'00'
GDS ISSUE ABEND	Free (state 12)	X'FF'	X'FF'

If CDBERR=X'00', the partner transaction has replied GDS ISSUE CONFIRMATION.

If the partner transaction replies GDS ISSUE ERROR, this is indicated by CDBERR (=X'FF') and the first two bytes of CDBERRCD=X'0889'. When the partner transaction replies GDS ISSUE ERROR in response to GDS SEND LAST CONFIRM, the LAST option is ignored and the conversation is *not* terminated. The conversation is switched to **receive state** (state 5).

If the partner transaction replies GDS ISSUE ABEND, both CDBERR and CDBFREE are both set (X'FF'), and the first two bytes of CDBERRCD contain X'0864'. The conversation is switched to **free state** (state 12).

How to synchronize conversations using SYNCPOINT commands

Data synchronization (SYNCPOINT and SYNCPOINT ROLLBACK) affects all connected conversations at sync level 2. The use of these commands in DTP is described in "Part 6. Syncpointing a distributed process" on page 117.

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. Once this has happened the conversation is unusable and **both** transactions must issue a GDS FREE command to release the session.

Normal termination of a conversation

The GDS SEND LAST command is used to terminate a conversation. It should be used in conjunction with either the WAIT or CONFIRM options or the SYNCPOINT command (depending on the conversation sync level). Table 30 on page 84 describes this.

Table 30. Terminating commands for different sync levels

Sync level	Command sequence
0	GDS SEND LAST WAIT GDS FREE
1	GDS SEND LAST CONFIRM GDS FREE
2	GDS SEND LAST[31] SYNCPOINT GDS FREE

Note: A distributed transaction should not end a conversation by issuing an EXEC CICS RETURN command, but instead follow the sequence of commands shown in Table 30. The issue of an EXEC CICS RETURN could lead to one or both transactions ending abnormally.

Emergency termination of a conversation

The GDS ISSUE ABEND command provides a means of abnormally ending the conversation. It is valid for all levels of synchronization, but should be avoided at sync level 2, because its use at the wrong time can lead to a loss of data integrity.

GDS ISSUE ABEND can be issued by either transaction, whether it is in send or receive state, at any time after the conversation has started. For a transaction in **send state** (state 2), any deferred data that is waiting for transmission is flushed before the GDS ISSUE ABEND command is transmitted.

The transaction that issues the GDS ISSUE ABEND command is not itself abended. It must, however, issue a FREE command for the conversation unless it is designed to terminate immediately.

If a GDS 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 a GDS ISSUE ABEND is received, both CDBERR and CDBFREE set (X'FF'), the first two bytes of CDBERRCD contain X'0864'. The only command that can be subsequently issued for the conversation is GDS FREE.

Unexpected termination of a conversation

If a partner systems fails or a session goes out of service in the middle of a DTP conversation, the conversation is terminated abnormally and the application informed the next time a command accesses the session. In addition, both CDBERR and CDBFREE are set on (X'FF'), and CDBERRCD contains one of the following values representing the reason for the error.

X'08640001' - partner system with persistent session support has failed and restarted
X'1008600B' - session has failed due to a protocol error
X'A0000100' - temporary session failure
X'A0010100' - RTIMOUT triggered.

31. It is important that the GDS SEND LAST command for sync level 2 is **not** accompanied by WAIT or CONFIRM because either of these options will cause the conversation to end before the subsequent syncpoint has propagated to the partner transaction. This may mean that protected resources of one transaction could be committed while those in the partner transaction could be backed out. The resulting state errors may also lead to the session being unbound.

Checking the outcome of GDS commands

The CICS exec interface block (EIB) is not affected by EXEC CICS GDS commands, and no CICS conditions can be raised when EXEC CICS GDS commands are executed. Instead, you must provide data areas in your application to receive return codes and session status information.

The data areas required are:

- A 6-byte area to receive RETCODE information
- A 24-byte area to receive CONVDATA information.

Within the bounds of the programming language you are using, you can give these areas any identifiers you like. They must be named explicitly in most EXEC CICS GDS commands.

Checking the response from a GDS command can be separated into three stages:

1. Testing for request failure; this involves testing RETCODE.
2. Testing for indicators received on the conversation. These indicators are found in CONVDATA.
3. Testing the conversation state.

Testing for request failure

The RETCODE area is used to detect any errors that occur when an EXEC CICS GDS command is executed. These errors correspond to CICS exception conditions, such as NOTALLOC, that can be raised when EXEC CICS commands are executed.

These errors usually reflect failure of the request. Figure 23 on page 86 shows the possible hexadecimal values for the first three bytes of RETCODE. These values are structured so that the first byte indicates the general error description and subsequent bytes provide the detail.

```
00 .. ..   Normal return code
01 .. ..   ALLOCATE failure (applicable only to GDS ALLOCATE)
01 04 ..     SYSBUSY, unknown modename, task cancelled
01 04 04       No bound contention winner available (SYSBUSY)
01 04 08       Modename not known on this system
01 04 0C       Attempt to use reserved modename SNASVCMG, or no COS
               table in VTAM for the modename
01 04 10       Task cancelled during queuing of ALLOCATE
01 04 14       The requested modegroup is closed
01 04 18       The requested modegroup is draining
01 08 ..     SYSID is out of service
01 08 00       Connection out of service or in quiesce state, no usable
               sessions in requested modegroup, or VTAM ACB is closed

01 08 04       Maximum number of queued ALLOCATE requests specified
               on QUEUELIMIT CONNECTION parameter exceeded
01 08 08       ALLOCATE queue purged because MAXQTIME would be exceeded

01 0C ..     SYSID is not known in TCT
01 0C 00       SYSID name is not known
01 0C 04       SYSID name is not that of an APPC connection
01 0C 14       NETNAME specified in PARTNER definition is not known
02 0C 00   PARTNER is not known
03 .. ..   INVREQ error
03 00 ..     Session is either not defined as APPC, in use by
             CPI Communications, or (for EXTRACT PROCESS) not
             the principal facility
03 04 ..     GDS command issued on a conversation that is not basic
03 08 ..     Command issued in wrong state
03 0C ..     Sync level cannot be supported or cannot support the
             command issued
03 10 ..     LL error on a GDS SEND
03 14 ..     SEND CONFIRM or ISSUE CONFIRMATION used at sync level 0
03 24 ..     GDS ISSUE PREPARE used in wrong state
04 .. ..   NOTALLOC error (CONVID specifies an unallocated session)
05 .. ..   LENGERR error (FLENGTH, MAXFLENGTH, PROCLENGTH, PIPLENGTH,
           or MAXPROCLEN error)
06 00 00   PROFILE specified in PARTNER definition is not known
```

Figure 23. RETCODE values

Testing indicators

When RETCODE shows a normal return code from a GDS command, the CONVDATA area (where applicable) contains information on the indicators received on the conversation. These indicators can be used to find out why the conversation state is what it is.

The structure of the CONVDATA area is shown in Table 31.

Table 31. Structure of the conversation data block

Field name	Length (bytes)	Meaning
CDBCOMPL	1	X'FF' = data complete
CDBSYNC	1	X'FF' = SYNCPOINT required
CDBFREE	1	X'FF' = FREE required
CDBRECV	1	X'FF' = RECEIVE required
CDBSIG	1	X'FF' = SIGNAL received

Table 31. Structure of the conversation data block (continued)

Field name	Length (bytes)	Meaning
CDBCONF	1	X'FF' = CONFIRM received
CDBERR	1	X'FF' = ERROR received
CDBERRCD	4	Error code (when CDBERR set)
CDBSYNRB	1	X'FF' = SYNCPOINT ROLLBACK required
CDBRSVD	12	Reserved

These definitions are provided in copybook DFHCDBLK. There is one copybook for C, which defines a *typedef* for the structure, and another copybook for assembler. To provide the flexibility to enable your application to manage more than one conversation at the same time, the assembler version does not contain a DSECT statement.

The meanings of the CONVDATA fields are as follows:

CDBERR
when set to X'FF' indicates an error has occurred on the conversation. The reason is in CDBERRCD. This could be as a result of a GDS ISSUE ERROR, GDS ISSUE ABEND, or SYNCPOINT ROLLBACK command issued by the partner transaction. CDBERR 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 CDBERR in conjunction with CDBFREE and CDBSYNRB after every GDS command.

CDBERRCD
contains the reason for CDBERR. If CDBERR is not set, this field is not used.

CDBFREE
when set to X'FF' indicates that the partner transaction had ended the conversation. It should be tested along with CDBERR and CDBSYNC to find out exactly how to end the conversation.

CDBSIG
when set to X'FF' indicates the partner transaction or system has issued and GDS ISSUE SIGNAL command.

CDBSYNRB
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 32 shows how these CDB fields interact.

Table 32. Interaction between some CDB fields—all DTP commands

CDB- ERR	CDB- FREE	CDB- SYNRB	CDBERRCD	Description
X'FF'	X'00'	X'00'	X'08890000' X'08890001'	The partner transaction has sent GDS ISSUE ERROR
X'FF'	X'00'	X'00'	X'08890100' X'08890101'	The partner system has sent GDS ISSUE ERROR
X'FF'	X'00'	X'00'	X'A0020000'	Error in data received from partner

Table 32. Interaction between some CDB fields—all DTP commands (continued)

CDB- ERR	CDB- FREE	CDB- SYNRB	CDBERRCD	Description
X'FF'	X'FF'	X'00'	X'08640000'	The partner transaction has sent GDS ISSUE ABEND
X'FF'	X'FF'	X'00'	X'08640001'	The partner system has sent GDS 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 triggered
X'FF'	X'FF'	X'00'	X'10086032'	The PIP data sent with the GDS CONNECT PROCESS was incorrectly specified
X'FF'	X'FF'	X'00'	X'10086034'	The partner system does not support basic 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 GDS 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 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 CONVDATA fields are relevant only to GDS RECEIVE commands:

CDBCOMPL
: 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 GDS RECEIVE LLID command.

CDBCONF
: when set to X'FF' indicates that the partner transaction has issued a GDS SEND CONFIRM command and requires a response.

CDBRECV
: is only used when CDBERR is not set. When CDRECV is on (X'FF'), another GDS RECEIVE is required.

CDBSYNC
: 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 33 shows how some of these CDB fields interact for RECEIVE commands.

Table 33. Interaction between some CDB fields—RECEIVE commands only

CDB- ERR	CDB- FREE	CDB- RECV	CDB- SYNC	CDB- CONF	Description
X'00'	X'00'	X'00'	X'00'	X'00'	The partner transaction or system has issued GDS SEND INVITE WAIT. The local program is now in send state.

Table 33. Interaction between some CDB fields—RECEIVE commands only (continued)

CDB- ERR	CDB- FREE	CDB- RECV	CDB- SYNC	CDB- CONF	Description
X'00'	X'00'	X'00'	X'FF'	X'00'	The partner transaction or system has issued GDS 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 GDS 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 GDS SEND or GDS 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 GDS 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 GDS 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 GDS 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 GDS SEND LAST CONFIRM. The local program is in conffree state.

After analyzing the CONVDATA fields, you can test the conversation state to find out which GDS commands you can issue next. See “Chapter 8. State transitions in APPC basic conversations” on page 93.

Checking CONVDATA fields and the conversation state

Most of the information supplied by the CONVDATA fields can also be obtained from the conversation state. However, although the conversation state is easier to test, you cannot ignore CDBERR (and CDBERRCD).

For example, if after a GDS SEND INVITE WAIT or a GDS RECEIVE command has been issued, the conversation is in **receive state** (state 5), only CDBERR indicates that the partner transaction has sent a GDS ISSUE ERROR. This is illustrated in Figure 24 on page 90 and Figure 25 on page 91.

It should be noted that the state tables provided contain not only conversation states and commands issued, but also relevant CONVDATA field settings. The order in which these fields are shown provides a sensible sequence of checks for an application.

```
*         ...
* Check return code from RECEIVE
          NC    WRETC,WRETC
          BNZ   BADRET         Request-related error, analyze
*         ...                  Request successful
          NC    CDBERR,CDBERR
          BNZ   ERROR          Error indicated, analyze
*         ...                  No errors, check state
          CLC   WSTATE,DFHVALUE(SYNCFREE)
          BE    OKSYNFR        Partner issued SYNCPOINT and LAST
          CLC   WSTATE,DFHVALUE(SYNCRECEIVE)
          BE    OKSYNRC        Partner issued SYNCPOINT
          CLC   WSTATE,DFHVALUE(SYNCSEND)
          BE    OKSYNSE        Partner issued SYNCPOINT and INVITE
          CLC   WSTATE,DFHVALUE(CONFFREE)
          BE    OKCONFR        Partner issued CONFIRM and LAST
          CLC   WSTATE,DFHVALUE(CONFRECEIVE)
          BE    OKCONRC        Partner issued CONFIRM
          CLC   WSTATE,DFHVALUE(CONFSEND)
          BE    OKCONSE        Partner issued CONFIRM and INVITE
          CLC   WSTATE,DFHVALUE(FREE)
          BE    OKFREE         Partner issued LAST
          CLC   WSTATE,DFHVALUE(SEND)
          BE    OKSEND         Partner issued INVITE
          CLC   WSTATE,DFHVALUE(RECEIVE)
          BE    OKRECV         Processing for receipt of data
*                              (including CDBCOMPL for incomplete data)

          B     LOGICERR       Logic error, should never happen
*         ...
ERROR     DS    0H
*                              Error indicated
          CLC   WSTATE,DFHVALUE(ROLLBACK)
          BE    ERRRLBK        ROLLBACK received
          CLC   WSTATE,DFHVALUE(FREE)
          BE    ERRFREE        ISSUE ABEND & TERMERR received,
*                              reason in CDBERRCD
          CLC   WSTATE,DFHVALUE(RECEIVE)
          BE    ERRRECV        ISSUE ERROR received, reason in CDBERRCD

          B     LOGICERR       Logic error, should never happen
*         ...
BADRET    DS    0H
*         ...                  Examine RETCODE for source of error
*         ...
WSTATE    DS    F
WRETC     DS    XL6
WCDB      DS    0CL24
          COPY  DFHCDBLK
*         ...
```

Figure 24. Checking the outcome of a GDS RECEIVE command

```
*          ...
* Check return code from SEND INVITE WAIT
           NC    WRETC,WRETC
           BNZ   BADRET         Request-related error, analyze RETCODE
*          ...                  Request successful
           NC    CDBERR,CDBERR
           BNZ   ERROR          Error indicated, analyze state
*          ...                  No errors, check state
           CLC   WSTATE,DFHVALUE(RECEIVE)
           BE    OKRECV         Processing for receipt of data
*                               (including CDBCOMPL for incomplete data)

           B     LOGICERR       Logic error, should never happen
*          ...
ERROR      DS    0H
*                               Error indicated
           CLC   WSTATE,DFHVALUE(ROLLBACK)
           BE    ERRRLBK        ROLLBACK received
           CLC   WSTATE,DFHVALUE(FREE)
           BE    ERRFREE        ISSUE ABEND & TERMERR received,
*                               reason in CDBERRCD
           CLC   WSTATE,DFHVALUE(RECEIVE)
           BE    ERRRECV        ISSUE ERROR received, reason in CDBERRCD

           B     LOGICERR       Logic error, should never happen
*          ...
BADRET     ...                  Examine RETCODE for source of error
*          ...
*
WSTATE     DS    F
WRETC      DS    XL6
WCDB       DS    0CL24
           COPY  DFHCDBLK
*          ...
```

Figure 25. Checking the outcome of a GDS SEND INVITE WAIT command

Summary of commands for APPC basic conversations

Table 34 shows the commands used in APPC basic conversations. For programming information about these commands, see the *CICS Application Programming Reference* manual.

Table 34. Summary of commands used in basic conversations

Use to ...	Sync levels	Command	Page
Acquire a session to the partner system.	0,1,2	GDS ALLOCATE	71
Initiate a conversation with a named process on the partner system.	0,1,2	GDS CONNECT PROCESS	72
Obtain the session and connection identifiers of the transaction’s principal facility.	0,1,2	GDS ASSIGN	73
Access session-related information in the attach header that initiated the transaction.	0,1,2	GDS EXTRACT PROCESS	73
Send data and control information to the conversation partner.	0,1,2	GDS SEND	75
Receive data from the conversation partner.	0,1,2	GDS RECEIVE	77

Table 34. Summary of commands used in basic conversations (continued)

Use to ...	Sync levels	Command	Page
Transmit any deferred data or control indicators.	0,1,2	GDS WAIT	75
Reply positively to GDS SEND CONFIRM.	1,2	GDS ISSUE CONFIRMATION	82
Prepare a conversation partner for syncpointing.	2	GDS ISSUE PREPARE	120
Inform the conversation partner of a program-detected error.	0,1,2	GDS ISSUE ERROR	82
Signal an unusual condition to the conversation partner, usually against the flow of data.	0,1,2	GDS ISSUE SIGNAL	80
Inform the conversation partner that the conversation should be abandoned.	0,1,2	GDS ISSUE ABEND	84
Free the session.	0,1,2	GDS FREE	83
Inform all a transaction's conversation partners that it is ready to commit its recoverable resources.	2	SYNCPOINT	119
Inform all a transaction's conversation partners that it wants to back out changes to recoverable resources.	2	SYNCPOINT ROLLBACK	120

Chapter 8. State transitions in APPC basic conversations

This chapter shows how the state changes when GDS commands are issued in APPC basic conversations. The state transitions are presented in the form of state tables showing which commands can be issued while the conversation is in any given state. The tables also show how the conversation state changes as a result of a command.

The state tables for APPC basic 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 occur and the CDB flags raised when a command is issued. CDB fields are used to return indicators from the conversation. They are described in "Checking the outcome of GDS commands" on page 85.

How to use the state tables

The commands you can issue, coupled with the CDB flags that can be set after execution, are shown in column 1 down the left side of the table. The possible conversation states are shown across the top of the table. The states correspond to the columns of the table. The intersection of a row (command and CDB flag) and a column (state) represents the state transition, if any, that occurs when a particular command, issued in a particular state, returns a particular CDB flag. The order in which the CDB flags appear with a command also shows the order in which you test the CDB flags in your program.

A number at an intersection indicates the next state. Other symbols represent other conditions, as follows:

Symbol	Meaning
N/A	Cannot occur.
×	The CDB flag is any one that has not been covered in earlier rows, or it is irrelevant (but see the note on CDBSIG if you want to use GDS ISSUE SIGNAL).
Ab	The command is not valid in this state. Issuing a command in a state in which it is not valid causes a bad response to be returned.
=	Remains in current state.
End	End of conversation.

Table 35. APPC basic conversations at sync level 0, part 1

Command issued	CDB flag returned[32]	ALLO-CATED[34] State 1	SEND State 2	PEND-RECEIVE State 3	PEND-FREE State 4	RECEIVE State 5	CONF-RECEIVE State 6
GDS CONNECT PROCESS[36]	EIBERR + EIBFREE	12	Ab	Ab	Ab	Ab	N/A
GDS CONNECT PROCESS[36]	×	2	Ab	Ab	Ab	Ab	N/A
GDS EXTRACT PROCESS[33]	×	=	=	=	=	=	N/A
GDS EXTRACT ATTRIBUTES	×	=	=	=	=	=	N/A
GDS SEND (any valid form)	CDBERR + CDBFREE	Ab	12	Ab	Ab	Ab	N/A
GDS SEND (any valid form)	CDBERR	Ab	5	Ab	Ab	Ab	N/A
GDS SEND INVITE WAIT	×	Ab	5	Ab	Ab	Ab	N/A
GDS SEND INVITE	×	Ab	3	Ab	Ab	Ab	N/A
GDS SEND LAST WAIT	×	Ab	12	Ab	Ab	Ab	N/A
GDS SEND LAST	×	Ab	4	Ab	Ab	Ab	N/A
GDS SEND WAIT	×	Ab	=	Ab	Ab	Ab	N/A
GDS SEND	×	Ab	=	Ab	Ab	Ab	N/A
GDS RECEIVE	CDBERR + CDBFREE	Ab	Ab	Ab	Ab	12	N/A
GDS RECEIVE	CDBERR	Ab	Ab	Ab	Ab	=	N/A
GDS RECEIVE	CDBFREE	Ab	Ab	Ab	Ab	12	N/A
GDS RECEIVE	CDBRECV	Ab	Ab	Ab	Ab	=	N/A
GDS RECEIVE LLID	CDBCOMPL	Ab	Ab	Ab	Ab	=	N/A
GDS RECEIVE	×	Ab	Ab	Ab	Ab	2	N/A
GDS ISSUE ERROR	CDBFREE	Ab	12	12	Ab	12	N/A
GDS ISSUE ERROR	×	Ab	=	2	Ab	2	N/A
GDS ISSUE ABEND	×	Ab	12	12	12	12	N/A
GDS ISSUE SIGNAL[35]	×	Ab	=	=	Ab	=	N/A
GDS WAIT	×	Ab	=	5	12	Ab	N/A
GDS FREE	×	End	Ab	Ab	End	Ab	N/A

Note: See page 100 for footnotes.

Table 36. APPC basic conversations at sync level 0, part 2

CONF-SEND	CONF-FREE	SYNC-RECEIVE	SYNC-SEND	SYNC-FREE	FREE	ROLL-BACK	
State 7	**State 8**	**State 9**	**State 10**	**State 11**	**State 12**	**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
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

Table 37. APPC basic conversations at sync level 1, part 1

Command issued	CDB flag returned[32]	ALLO-CATED[34] State 1	SEND State 2	PEND-RECEIVE State 3	PEND-FREE State 4	RECEIVE State 5	CONF-RECEIVE State 6
GDS CONNECT PROCESS[36]	EIBERR + EIBFREE	12	Ab	Ab	Ab	Ab	Ab
GDS CONNECT PROCESS[36]	×	2	Ab	Ab	Ab	Ab	Ab
GDS EXTRACT PROCESS[33]	×	=	=	=	=	=	=
GDS EXTRACT ATTRIBUTES	×	=	=	=	=	=	=
GDS SEND (any valid form)	CDBERR + CDBFREE	Ab	12	Ab	12	Ab	Ab
GDS SEND (any valid form)	CDBFREE	Ab	12	Ab	Ab	Ab	Ab
GDS SEND INVITE WAIT	×	Ab	5	Ab	Ab	Ab	Ab
GDS SEND INVITE CONFIRM	×	Ab	5	Ab	Ab	Ab	Ab
GDS SEND INVITE	×	Ab	3	Ab	Ab	Ab	Ab
GDS SEND LAST WAIT	×	Ab	12	Ab	Ab	Ab	Ab
GDS SEND LAST CONFIRM	×	Ab	12	Ab	Ab	Ab	Ab
GDS SEND LAST	×	Ab	4	Ab	Ab	Ab	Ab
GDS SEND WAIT	×	Ab	=	Ab	Ab	Ab	Ab
GDS SEND CONFIRM	×	Ab	=	5[37]	12[37]	Ab	Ab
GDS SEND	×	Ab	=	Ab	Ab	Ab	Ab
GDS RECEIVE	CDBERR + CDBFREE	Ab	Ab	Ab	Ab	12	Ab
GDS RECEIVE	CDBERR	Ab	Ab	Ab	Ab	=	Ab
GDS RECEIVE	CDBCONF + CDBFREE	Ab	Ab	Ab	Ab	8	Ab
GDS RECEIVE	CDBCONF + CDBRECV	Ab	Ab	Ab	Ab	6	Ab
GDS RECEIVE	CDBCONF	Ab	Ab	Ab	Ab	7	Ab
GDS RECEIVE	CDBFREE	Ab	Ab	Ab	Ab	12	Ab
GDS RECEIVE	CDBRECV	Ab	Ab	Ab	Ab	=	Ab
GDS RECEIVE LLID	CDBCOMPL	Ab	Ab	Ab	Ab	=	Ab
GDS RECEIVE	×	Ab	Ab	Ab	Ab	2	Ab
GDS ISSUE CONFIRMATION	×	Ab	Ab	Ab	Ab	Ab	5
GDS ISSUE ERROR	CDBFREE	Ab	12	12	Ab	12	12
GDS ISSUE ERROR	×	Ab	=	2	Ab	2	2
GDS ISSUE ABEND	×	Ab	12	12	12	12	12
GDS ISSUE SIGNAL[35]	×	Ab	=	=	Ab	=	=
GDS WAIT	×	Ab	=	5	12	Ab	Ab
GDS FREE	×	End	Ab	Ab	End	Ab	Ab

Note: See page 100 for footnotes.

CONF-SEND	CONF-FREE	SYNC-RECEIVE	SYNC-SEND	SYNC-FREE	FREE	ROLL-BACK	
State 7	**State 8**	**State 9**	**State 10**	**State 11**	**State 12**	**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
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 38. APPC basic conversations at sync level 2, part 1

Command issued	CDB flag returned[32]	ALLO-CATED[34] State 1	SEND State 2	PEND-RECEIVE State 3	PEND-FREE State 4	RECEIVE State 5	CONF-RECEIVE State 6
GDS CONNECT PROCESS[36]	EIBERR + EIBFREE	12	Ab	Ab	Ab	Ab	Ab
GDS CONNECT PROCESS[36]	×	2	Ab	Ab	Ab	Ab	Ab
GDS EXTRACT PROCESS[33]	×	=	=	=	=	=	=
GDS EXTRACT ATTRIBUTES	×	=	=	=	=	=	=
GDS SEND (any valid form)	CDBERR + CDBFREE	Ab	12	Ab	12	Ab	Ab
GDS SEND (any valid form)	CDBERR	Ab	5	Ab	12	Ab	Ab
GDS SEND INVITE WAIT	×	Ab	5	Ab	Ab	Ab	Ab
GDS SEND INVITE CONFIRM	×	Ab	5	Ab	Ab	Ab	Ab
GDS SEND INVITE	×	Ab	3	Ab	Ab	Ab	Ab
GDS SEND LAST WAIT[38]	×	Ab	12	Ab	Ab	Ab	Ab
GDS SEND LAST CONFIRM[38]	×	Ab	12	Ab	Ab	Ab	Ab
GDS SEND LAST	×	Ab	4	Ab	Ab	Ab	Ab
GDS SEND WAIT	×	Ab	=	Ab	Ab	Ab	Ab
GDS SEND CONFIRM	×	Ab	=	5	12[37]	Ab	Ab
GDS SEND	×	Ab	=	Ab	Ab	Ab	Ab
GDS RECEIVE	CDBERR + CDBSYNRB	Ab	Ab	Ab	Ab	13	Ab
GDS RECEIVE	CDBERR + CDBFREE	Ab	Ab	Ab	Ab	12	Ab
GDS RECEIVE	CDBERR	Ab	Ab	Ab	Ab	=	Ab
GDS RECEIVE	CDBSYNC + CDBFREE	Ab	Ab	Ab	Ab	11	Ab
GDS RECEIVE	CDBSYNC + CDBRECV	Ab	Ab	Ab	Ab	9	Ab
GDS RECEIVE	CDBSYNC	Ab	Ab	Ab	Ab	10	Ab
GDS RECEIVE	CDBCONF + CDBFREE	Ab	Ab	Ab	Ab	8	Ab
GDS RECEIVE	CDBCONF + CDBRECV	Ab	Ab	Ab	Ab	6	Ab
GDS RECEIVE	CDBCONF	Ab	Ab	Ab	Ab	7	Ab
GDS RECEIVE	CDBFREE	Ab	Ab	Ab	Ab	12	Ab
GDS RECEIVE	CDBRECV	Ab	Ab	Ab	Ab	=	Ab
GDS RECEIVE LLID	CDBCOMPL	Ab	Ab	Ab	Ab	=	Ab
GDS RECEIVE	×	Ab	Ab	Ab	Ab	2	Ab

Note: See page 100 for footnotes.

Table 39. APPC basic conversations at sync level 2, part 2

CONF-SEND	CONF-FREE	SYNC-RECEIVE	SYNC-SEND	SYNC-FREE	FREE	ROLL-BACK	
State 7	**State 8**	**State 9**	**State 10**	**State 11**	**State 12**	**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 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
Ab	Ab	Ab	Ab	Ab	Ab	Ab	After confirm flow detected
Ab	Ab	Ab	Ab	Ab	Ab	Ab	After error flow detected
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

table continued ...

Table 40. APPC basic conversations at sync level 2, part 3

Command issued	CDB flag returned[32]	ALLO-CATED[34] State 1	SEND State 2	PEND-RECEIVE State 3	PEND-FREE State 4	RECEIVE State 5	CONF-RECEIVE State 6
GDS ISSUE CONFIRMATION	×	Ab	Ab	Ab	Ab	Ab	5
GDS ISSUE ERROR	CDBFREE	Ab	12	12	Ab	12	12
GDS ISSUE ERROR	×	Ab	=	2	Ab	2	2
GDS ISSUE ABEND	×	Ab	12	12	12	12	12
GDS ISSUE SIGNAL	×	Ab	=	=	Ab	=	=
GDS ISSUE PREPARE	CDBERR + CDBSYNRB	Ab	13	13	13	Ab	Ab
GDS ISSUE PREPARE	CDBERR + CDBFREE	Ab	12	12	12	Ab	Ab
GDS ISSUE PREPARE	CDBERR	Ab	5	5	5	Ab	Ab
GDS ISSUE PREPARE	×	Ab	10	9	11	Ab	Ab
SYNCPOINT[40]	EIBRLDBK	=	2 or 5[39]	2 or 5[39]	2 or 5[39]	Ab	Ab
SYNCPOINT[40]	×	=	=	5	12	Ab	Ab
SYNCPOINT ROLLBACK[40]	×	=	2 or 5[39]	2 or 5[39]	2 or 5[39]	2 or 5[39]	2 or 5[39]
GDS WAIT	×	Ab	=	5	12	Ab	Ab
GDS FREE	×	End	Ab	Ab	End	Ab	Ab

32. CDBSIG 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 occur at any time after every command that affects the CDB flags. However, used for the purpose for which it was intended, it usually occurs after a GDS SEND command. Its priority in the order of testing depends on the role you give it in the application.

33. You can issue the GDS EXTRACT PROCESS command from the back-end transaction transaction only.

34. Before a session is allocated, there is no conversation, and therefore no conversation state. The GDS ALLOCATE command does not appear in the tables. This is because each GDS ALLOCATE gets a session to start a new conversation and does not affect any conversation that is already in progress. After GDS ALLOCATE is successful, the front-end transaction starts the new conversation in **allocated state**.

35. GDS ISSUE SIGNAL sets the partner transaction's CDBSIG flag.

36. The back-end transaction starts in RECEIVE state after the front-end transaction has issued GDS CONNECT PROCESS.

37. No data may be included with GDS SEND CONFIRM.

38. Although CICS allows you to terminate a sync level-2 conversation using the GDS SEND LAST WAIT or GDS SEND LAST CONFIRM commands, doing this deviates from the APPC architecture and should be avoided. See "CICS deviations from the APPC architecture" on page 161.

Table 41. APPC basic conversations at sync level 2, part 4

CONF-SEND	CONF-FREE	SYNC-RECEIVE	SYNC-SEND	SYNC-FREE	FREE	ROLL-BACK	
State 7	**State 8**	**State 9**	**State 10**	**State 11**	**State 12**	**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
=	=	=	=	=	Ab	Ab	Immediately
Ab	Ab	Ab	Ab	Ab	Ab	Ab	After response from partner
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 response from partner
Ab	Ab	2 or 5[39]	2 or 5[39]	2 or 5[39]	=	Ab	After response from partner
Ab	Ab	5	2	12	=	Ab	After response from partner
2 or 5[39]	2 or 5[39]	2 or 5[39]	2 or 5[39]	2 or 5[39]	Ab	2 or 5[39]	After rollback across UOW
Ab	Ab	Ab	Ab	Ab	Ab	Ab	Immediately
Ab	Ab	Ab	Ab	Ab	End	Ab	Immediately

39. The state of each conversation after rollback depends on several factors:
    - The system you are communicating with. Some earlier versions of CICS handle rollback differently from CICS Transaction Server for z/OS, Version 2 Release 2.
    - 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 Transaction Server for z/OS, Version 2 Release 2 follows the architecture.

    A conversation may be in **free state** after rollback if the it has been terminated in one of these ways:
    - Abnormally due to session failure or deallocate abend being received
    - Because the partner transaction has issued a GDS 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.
40. 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, including MRO conversations.

Initial states

The front-end transaction in a conversation must issue a GDS 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 an application to inquire on the current conversation state. The first is to use the EXEC CICS GDS EXTRACT ATTRIBUTES STATE command and the second is to use the STATE parameter on the GDS commands. In both cases the current state is returned to the application in a CICS value data area (cvda). Table 42 shows how the cvda codes relate to the conversation state. The table also shows the symbolic names defined for the cvda values.

Table 42. 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
Syncreceive	9	DFHVALUE(SYNCRECEIVE)	92
Syncsend	10	DFHVALUE(SYNCSEND)	93
Syncfree	11	DFHVALUE(SYNCFREE)	91
Free	12	DFHVALUE(FREE)	85
Rollback	13	DFHVALUE(ROLLBACK)	89

Part 5. Writing programs for LUTYPE6.1 conversations

This is the last of four parts detailing the CICS APIs available for DTP programming.

- “Part 2. Writing programs for APPC mapped conversations” on page 21
- “Part 3. Writing programs for MRO conversations” on page 51
- “Part 4. Writing programs for APPC basic conversations” on page 69
- Part 5. Writing programs for LUTYPE6.1 conversations.

The different APIs are compared in “Part 1. Concepts and design considerations” on page 1.

Part 5 contains:

- “Chapter 9. LUTYPE6.1 conversation flow” on page 105.

  This uses CICS-to-IMS communication as the basis for discussing LUTYPE6.1 DTP programming.
- “Chapter 10. State transitions in LUTYPE6.1 conversations” on page 113. This discusses the state transitions that occur when transactions use LUTYPE6.1 conversations under the EXEC CICS API. State transitions are presented in the form of a state table showing which commands can be issued while the conversation is in any given state. The state table also shows how the conversation state changes as a result of issuing a command.

Chapter 9. LUTYPE6.1 conversation flow

This chapter introduces some of the DTP commands for LUTYPE6.1 conversation flow. 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 chapter contains the following topics:
- Starting the conversation
- “Transferring data on the conversation” on page 106
- “Ending the conversation” on page 108
- “Checking the outcome of a DTP command” on page 108
- “Considerations for the front-end transaction” on page 110
- “Summary of commands for LUTYPE6.1 conversations” on page 112.

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 final subsection discusses conversation initiation failure.

Conversation initiation

The front-end transaction is responsible for acquiring a session, specifying the conversation characteristics, and requesting the startup of the back-end transaction in the partner system.

Allocating a session to the conversation

Initially, there is no conversation, and therefore no conversation state. The front-end transaction acquires a session to start a new conversation by issuing an ALLOCATE command.

The RESP value should be checked to ensure that a session has been allocated. If successful, the RESP value is DFHRESP(NORMAL), the conversation is in **allocated state** (state 1) and the session identifier (**convid**) from EIBRSRCE must be saved immediately. The convid must be used in subsequent commands for this conversation.

If the front-end transaction is started by ATI in the local system, and is required to hold a conversation with an LUTYPE6.1 session as its principal facility, the session has already been allocated when the transaction starts. You can omit the SESSION option from commands relating to the principal facility. If, however, you want to name the session explicitly in these commands, you should obtain its name from EIBTRMID.

Connecting the partner transaction

When a session has been acquired, the next step is to cause the partner transaction to be initiated. The state table shows that, in **allocated state** (state 1), one of the commands available is SEND. Using this command, the back-end transaction identifiers can be specified in the first four bytes of the data which, when transferred to the partner system, will attach the required back-end transaction. The send buffer containing the transaction name together with any other data, will be flushed immediately and the front-end transaction will wait until a response is received from the back-end transaction.

Alternatively, when a session has been acquired, the front-end transaction can build and send an attach header with the first transmission of data. The attach header can be built using the BUILD ATTACH command.

When using the BUILD ATTACH command, you must give a name to the built attach header which can then be used in the ATTACHID option of the first SEND (or converse) command. The back-end transaction name should also be specified.

Back-end transaction initiation

The back-end transaction is initiated either by an attach header received from the partner system or by a transaction name included in the incoming data, and is started with the session as its principal facility. Initially, the back-end transaction should determine the convid from EIBTRMID. This is not strictly necessary because the session is the back-end transaction's principal facility making the CONVID parameter optional for DTP commands on this conversation. However, the convid is very useful for audit trails. Also, if the back-end transaction is involved in more than one conversation, then always specifying the convid improves program readability and problem determination.

A CICS transaction can be the back-end transaction in CICS-to-IMS communication only in the special case of SEND/RECEIVE asynchronous processing. The transaction is initiated by an LUTYPE6.1 attach FMH received from the remote IMS system, and is allowed to issue a single RECEIVE command only, possibly followed by an EXTRACT ATTACH command.

What happens if the back-end transaction fails to start

It is possible that the back-end transaction may fail to start up. This will result in the front-end transaction abending.

Transferring data on the conversation

This section discusses how to pass data between the front-end 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.

Sending data to the partner transaction

The SEND command is used to send data to the connected partner. This command is valid in allocated state (state 1) or **send state** (state 2). Because a successful simple SEND completes in **send state** (state 2), it is possible to issue a number of successive sends.

Switching from sending to receiving data

The column for send state (state 2) in the state table shows that there is more than one way of switching from **send state** (state 2) to **receive state** (state 5).

One possibility is to use a SEND INVITE command. The state table shows that after SEND INVITE the conversation switches to **pendreceive state** (state 3). As the column for state 3 shows, a WAIT TERMINAL command switches the conversation to **receive state** (state 5).

Another possibility is to specify INVITE and WAIT on the SEND command. As the state table shows, SEND INVITE WAIT switches the conversation to **receive state** (state 5).

Receiving data from the partner transaction

The RECEIVE command is used to receive data from the connected partner. The rows in the state tables for the RECEIVE command show the EIB fields that should be tested after issuing a RECEIVE command. As well as showing which field should be tested, the state tables also shows the order in which the tests should be made. Note that you should always test for RESP values.

The transaction whose side of the conversation is in **receive state** cannot change to **send state**, but can request a change of direction by using the ISSUE SIGNAL command. This causes the SIGNAL condition to be raised in the partner transaction the next time it issues a SEND, RECEIVE, or CONVERSE command. The application is responsible for determining the purpose of the SIGNAL condition and responding appropriately.

Waiting for a signal

A transaction can wait for its partner to send a signal. This is done by issuing the WAIT SIGNAL command and testing for the SIGNAL condition. The WAIT SIGNAL command suspends the transaction until its partner responds with an ISSUE SIGNAL command. This response activates the suspended transaction and raises the SIGNAL condition.

Combining sending and receiving

The CONVERSE command combines the functions SEND INVITE and RECEIVE. This command is useful when one transaction needs a response from the partner transaction to continue processing.

Communicating errors across a conversation

If a transaction is receiving data on a conversation and needs to notify its partner of an error, it can use the ISSUE SIGNAL command to request that the partner does a SEND INVITE. When the ISSUE SIGNAL request is received, EIBSIG is set to X'FF' and the SIGNAL condition is raised. Note that when a *signal* is received, the transaction is not obliged to issue SEND INVITE.

Safeguarding data integrity

If it is important to safeguard data integrity across connected transactions, then the following synchronization commands are available:

SYNCPOINT SRRCMIT (SAA verb for SYNCPOINT)

The use of these commands in DTP is described in "Chapter 11. Syncpointing a distributed process" on page 119.

Ending the conversation

The following sections describe the different ways a conversation can end, either unexpectedly or under transaction control. When under transaction control, one transaction will issue a request for termination and the other will receive this request. Once this has happened the conversation is unusable and **both** transactions must issue a FREE command to release the session.

Ending a conversation normally

The SEND LAST command is used to terminate a conversation. It should be used in conjunction with either the WAIT option or the SYNCPOINT command, and followed by the FREE command. However, SEND LAST WAIT will cause the conversation to end before the subsequent syncpoint can be propagated to the partner transaction. This may mean that the protected resources in one system could be committed whilst those in the other system could be backed out.

From the state table it can be seen that it is possible to end a conversation by issuing the FREE command provided the conversation is in **send state** (state 2). This will generate an implicit SEND LAST WAIT command before the FREE is executed and is therefore not recommended.

Note: A distributed transaction should not end a conversation by issuing an EXEC CICS RETURN command, but instead follow the sequence of commands described above. The issue of an EXEC CICS RETURN could lead to one or both transactions ending abnormally.

Unexpected termination of a conversation

From time to time, partner systems do fail and sessions go out of service. If this happens in the middle of a DTP conversation, the transaction will be terminated abnormally.

Checking the outcome of a DTP command

Checking the response from a DTP command can be separated into two stages:

1. Testing for request failure
2. Testing for indicators received on the conversation.

Testing for request failure is the same as for other EXEC CICS commands in that conditions are raised and may be handled using HANDLE CONDITION or RESP. EIBRCODE will also contain an error code.

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. (See the *CICS Application Programming Reference* manual for programming information on the contents and format of EIB fields.)

EIBFREE
when set to X'FF' indicates that the partner transaction has ended the conversation. It should be tested in conjunction with EIBSYNC to determine exactly how to end the conversation.

EIBSYNC
when set to X'FF' indicates the partner transaction/system has requested a syncpoint.

Table 43 shows how these EIB fields interact.

Table 43. Interaction of some EIB fields

EIB- FREE	EIB- SYNC	Description
X'FF'	X'00'	The partner transaction or system has sent SEND LAST followed by a FREE command.
X'FF'	X'FF'	The partner transaction or system has issued SEND LAST followed by SYNCPOINT. The local program should reply with a SYNCPOINT command followed by a FREE command.
X'00'	X'FF'	The partner transactions or system has issued a SYNCPOINT.

In addition, there is a group of EIB fields that are relevant only to the RECEIVE and CONVERSE commands. These are:

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.

EIBRECV
when set to X'FF' indicates the partner transaction did not use the INVITE option on its last SEND command.

EIBATT
when set to X'FF' indicates that the data received contained an attach header. The attach header is not passed to the application; however, EIBATT indicates that an EXTRACT ATTACH command is appropriate.

EIBFMH
when set to X'FF' indicates that the data passed to the application contains a concentrated FMH. This happens only when the partner CICS transaction builds an FMH in the data and the FMH option on the SEND command is specified.

Note: Profiles specifying INBFMH (ALL) must be used in the ALLOCATE commands if FMHs are to be sent and received and EIBATT or EIBFMH to be sent appropriately. The default profile DFHCICSA used for the session allocated by the front-end transaction, has INBFMH (ALL) specified. However, the default principal facility profile DFHCICST used for the back-end transaction does not have INBFMH (ALL) specified.

Considerations for the front-end transaction

Except in the special case of the receiving transaction in SEND/RECEIVE asynchronous processing, the CICS transaction is always the front-end transaction in CICS-to-IMS DTP.

The front-end transaction is responsible for acquiring a session to the remote IMS™ system and initiating the partner transaction.

Thereafter, the two transactions become equals. However, the front-end transaction is usually designed as the client, or driving, transaction.

Session allocation

You acquire an LUTYPE6.1 session to a remote IMS system by means of the ALLOCATE command, which has the following format:

```
ALLOCATE {SYSID(name)|SESSION(name)}
  [PROFILE(name)]
  [NOQUEUE]
```

You can use the SESSION option to request the use of a specific session to the remote IMS system, or you can use the SYSID option to name the partner system and allow CICS to select an available session. The use of the SESSION option is not normally recommended, because it can result in an application program queuing on a specific session when others are available. In most cases, therefore, you use the SYSID option to name the system with which the session is required.

If CICS cannot find the named system, or all sessions to that system are out of service, it raises the SYSIDERR condition. If CICS cannot find the named session, or that session is out of service, it raises the SESSIONERR condition.

The PROFILE option allows you to select a specified communication profile for an LUTYPE6.1 session. The profile, which is set up during resource definition, contains a set of terminal control processing options that are to be used for the session.

If you omit the PROFILE option, CICS uses the default profile DFHCICSA. This profile specifies INBFMH(ALL), which means that incoming function management headers are passed to your program and cause the INBFMH condition to be raised.

The NOQUEUE option allows you to specify explicitly that you do not want your request for a session to be queued if a session is not available immediately. A session is “not immediately available” in any of the following situations:

- All the sessions to the specified system are in use.
- The only available sessions are not bound (in which case CICS would have to bind a session).
- The only available sessions are contention losers (in which case CICS would have to bid to begin a bracket).

The action taken by CICS if a session is not immediately available depends on whether you specify NOQUEUE and also on whether your application has executed a HANDLE command for the SYSBUSY condition. The possible combinations are shown below:

- HANDLE for SYSBUSY condition
  - Control is returned immediately to the label specified in the HANDLE command, whether or not you have specified NOQUEUE.
- No HANDLE for SYSBUSY condition
  - If you have specified NOQUEUE, control is returned immediately to your application program. A RESP value of DFHRESP(SYSBUSY) is returned. You should test this field immediately after issuing the ALLOCATE command.
  - If you have omitted the NOQUEUE option, CICS queues the request until a session is available.

Whether a delay in acquiring a session is acceptable is dependent on your application.

Similar considerations apply to an ALLOCATE command that specifies SESSION rather than SYSID. The associated condition is SESSBUSY.

The session identifier

When a session has been allocated, the name by which it is known is available in the EIBRSRCE field in the EIB. Because EIBRSRCE will probably be overwritten by the next EXEC CICS command, you must acquire the session name immediately. It is the name that you must use in the SESSION option of all subsequent commands that relate to this session.

Summary of commands for LUTYPE6.1 conversations

Table 44 shows the commands used in LUTYPE6.1 conversations. For programming information about CICS commands, see the *CICS Application Programming Reference* manual.

Table 44. Summary of commands used in LUTYPE6.1 conversations

Use to ...	Command	Page
Acquire a session.	ALLOCATE	105
Build an attach header.	BUILD ATTACH	105
Access session-related information.	EXTRACT ATTACH	106
Send data and control information to the conversation partner.	SEND	106
Receive data from the conversation partner.	RECEIVE	107
Send and receive data on the conversation.	CONVERSE	107
Inform all partners of readiness to commit recoverable resources.	SYNCPOINT	119
Signal an unusual condition to the conversation partner, usually against the flow of data.	ISSUE SIGNAL	107
Suspend processing until the SIGNAL condition is raised.	WAIT SIGNAL	107
Ensure that CICS has transmitted any accumulated data or data flow control indicators before further processing.	WAIT TERMINAL	106
Free the session.	FREE	108

Chapter 10. State transitions in LUTYPE6.1 conversations

This chapter shows the state transitions that occur when transactions engage in LUTYPE6.1 conversations. The state transitions are presented in the form of a state table. The state table shows which commands a transaction can issue while the conversation is in any given state. It also shows how the conversation state changes as a result of any command.

The state table for LUTYPE6.1 conversations

The state table provides the following information for writing a DTP program. Firstly, it shows which commands can be issued from each conversation state. Secondly, it shows the results of issuing a command in terms of state transactions and EIB fields.

How to use the state table

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 the table. 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.
×	The EIB flag is any one that has not been covered in earlier rows, or it is irrelevant.
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.

Table 45. LUTYPE6.1 conversations, part 1

Command issued	EIB flag returned	ALLO-CATED[44] State 1	SEND State 2	PEND-RECEIVE State 3	PEND-FREE State 4	RECEIVE State 5	CONF-RECEIVE State 6
BUILD ATTACH	×	=	=	=	=	=	N/A
EXTRACT ATTACH	×	=	=	=	=	=	N/A
SEND INVITE WAIT	×	5	5	Ab	Ab	Ab	N/A
SEND INVITE	×	3	3	Ab	Ab	Ab	N/A
SEND LAST WAIT	×	12	12	Ab	Ab	Ab	N/A
SEND LAST	×	4	4	Ab	Ab	Ab	N/A
SEND	×	=	=	Ab	Ab	Ab	N/A
RECEIVE	EIBSYNC + EIBFREE	Ab	11	11	Ab	11	N/A
RECEIVE	EIBSYNC + EIBRECV	Ab	9	9	Ab	9	N/A
RECEIVE	EIBSYNC	Ab	10	10	Ab	10	N/A
RECEIVE	EIBFREE	Ab	12	12	Ab	12	N/A
RECEIVE	EIBRECV	Ab	5	5	Ab	=	N/A
RECEIVE NOTRUNCATE[41]	EIBCOMPL[41]	Ab	5	5	Ab	=	N/A
RECEIVE	×	Ab	2	2	Ab	2	N/A
CONVERSE[42]	EIB flags and states as for RECEIVE but allowed in send state						
ISSUE SIGNAL[45]	×	Ab	=	=	=	=	N/A
WAIT SIGNAL	×	Ab	=	=	=	=	N/A
SYNCPOINT[46]	×	=	=	5	12	Ab	N/A
WAIT TERMINAL	×	=	=	5	12	=	N/A
FREE	×	End[43]	End[43]	Ab	End	Ab	N/A

41. 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 small to contain all the data received, CICS truncates the data and sets the LENGERR condition.

42. Equivalent to:

```
SEND INVITE WAIT [FROM]
RECEIVE
```

43. Equivalent to:

```
SEND LAST WAIT
FREE
```

Table 46. LUTYPE6.1 conversations, part 2

CONF-SEND	CONF-FREE	SYNC-RECEIVE	SYNC-SEND	SYNC-FREE	FREE	ROLL-BACK	
State 7	**State 8**	**State 9**	**State 10**	**State 11**	**State 12**	**State 13**	**Command returns**
N/A	N/A	N/A	=	=	=	=	Immediately
N/A	N/A	=	=	=	=	N/A	Immediately
N/A	N/A	Ab	Ab	Ab	Ab	N/A	After data and CD flows
N/A	N/A	Ab	Ab	Ab	Ab	N/A	After data buffered
N/A	N/A	Ab	Ab	Ab	Ab	N/A	After data and EB flows
N/A	N/A	Ab	Ab	Ab	Ab	N/A	After data buffered
N/A	N/A	Ab	Ab	Ab	Ab	N/A	After data buffered
N/A	N/A	Ab	Ab	Ab	Ab	N/A	After sync flow detected
N/A	N/A	Ab	Ab	Ab	Ab	N/A	After sync flow detected
N/A	N/A	Ab	Ab	Ab	Ab	N/A	After sync flow detected
N/A	N/A	Ab	Ab	Ab	Ab	N/A	After EB detected
N/A	N/A	Ab	Ab	Ab	Ab	N/A	When data available
N/A	N/A	Ab	Ab	Ab	Ab	N/A	When data available
N/A	N/A	Ab	Ab	Ab	Ab	N/A	When data available
States as for RECEIVE							When data available
N/A	N/A	=	=	=	Ab	N/A	Immediately
N/A	N/A	=	=	=	Ab	N/A	After response from partner
N/A	N/A	5	2	12	=	N/A	After response from partner
N/A	N/A	Ab	Ab	Ab	Ab	N/A	Immediately
N/A	N/A	Ab	Ab	Ab	End	N/A	Immediately

44. Before a session is allocated, there is no conversation, and therefore no conversation state. The EXEC CICS ALLOCATE command does not appear in the table. 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**.

    You select the partner transaction program by issuing a SEND command or a CONVERSE command. You have the choice of identifying the transaction program either in the first four bytes of the user data or in the attach function management header built by the BUILD ATTACH command.

    The back-end transaction starts in RECEIVE state.

45. ISSUE SIGNAL sets the partner's EIBSIG flag.

46. The SYNCPOINT command does not relate to any particular conversation. It is propagated on all the conversations that are currently active for the task, including APPC and MRO conversations. All these conversations must be in send state or pendfree state.

Initial states

A front-end transaction can be initiated either from a transaction or by automatic transaction initiation (ATI).

A terminal-initiated front-end transaction 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 front-end transaction started by ATI in the local system, with an LUTYPE6.1 session as its principal facility, already has a session allocated. Such a transaction does not issue an ALLOCATE command, and its side of the conversation starts in **send state** (state 2).

A back-end transaction is initially in **receive state** (state 5).

Testing the conversation state

There is no way for an application to check the conversation state directly. The application must instead check RESP and the EIB fields after each command, and must follow the rules shown in the state table.

Part 6. Syncpointing a distributed process

This part discusses how to add syncpointing to a distributed process. The material concentrates on the programming aspects of using the EXEC CICS SYNCPOINT [ROLLBACK][47] command across APPC conversations at sync level 2 and MRO conversations.

The information in this part is presented in a single chapter:

"Chapter 11. Syncpointing a distributed process" on page 119.

47. The SAA equivalents for this syncpointing command (SRRCMIT and SRRBACK) are described in the *SAA Common Programming Interface Resource Recovery Reference* manual.

Chapter 11. Syncpointing a distributed process

This chapter discusses how to include syncpointing in a distributed process. It concentrates on the programming aspects of using the EXEC CICS SYNCPOINT [ROLLBACK][48] command across APPC conversations (basic and mapped) at sync level 2 and MRO conversations. 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 chapter contains the following topics:

- The SYNCPOINT command
- "The ISSUE PREPARE command" on page 120
- "The SYNCPOINT ROLLBACK command" on page 120
- "When a backout is required" on page 121
- "Synchronizing two CICS systems" on page 121
- "Synchronizing three or more CICS systems" on page 136
- "What really flows between APPC systems" on page 141.

The SYNCPOINT command

The SYNCPOINT command is used to commit recoverable resources. In a DTP environment, the effect of the SYNCPOINT command is propagated across all conversations using sync level 2 or MRO. So, no matter how many DTP transactions are connected by conversations at sync level 2, the distributed process should be designed such 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 (CDBSYNC in the APPC basic interface) after issuing a 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 (X'FF'), which must be tested on return from the SYNCPOINT command. (This is also true for APPC basic conversations.)

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 SYNCPOINT ROLLBACK command" on page 120.

48. The SAA equivalents for this syncpointing command (SRRCMIT and SRRBACK) are described in the *SAA Common Programming Interface Resource Recovery Reference* manual.

Examples of these commands are given in "Synchronizing two CICS systems" on page 121 and "Synchronizing three or more CICS systems" on page 136.

The ISSUE PREPARE command

The ISSUE PREPARE (GDS ISSUE PREPARE for the APPC basic interface) 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 SYNCPOINT ROLLBACK command

The 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 MRO or 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 (CDBERR and CDBSYNRB in the basic interface) are set.

The rules for determining the state after SYNCPOINT ROLLBACK depend on the CICS release of the partner system. If the partner system is a release earlier than CICS/ESA 3.2.1, the rollback initiator completes backout processing in **send state** (state 2), and the partner completes in **receive state** (state 5). If the partner system is CICS/ESA 3.2.1 or later, the conversation state of each partner is restored to the state at the beginning of the distributed unit of work.

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 can check the conversation state by using the STATE option on the command following SYNCPOINT ROLLBACK. However, to avoid the possibility of an abend, you are recommended to follow each SYNCPOINT ROLLBACK command with an EXTRACT ATTRIBUTES STATE command instead.

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 (CDBERR and CDBFREE in the basic interface) 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 (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 MRO or sync level 2.

The examples illustrate the following scenarios:

- "SYNCPOINT in response to SYNCPOINT"
- "SYNCPOINT in response to ISSUE PREPARE" on page 124
- "SYNCPOINT ROLLBACK in response to SYNCPOINT ROLLBACK" on page 125
- "SYNCPOINT ROLLBACK in response to SYNCPOINT" on page 126
- "SYNCPOINT ROLLBACK in response to ISSUE PREPARE" on page 127
- "ISSUE ERROR in response to SYNCPOINT" on page 128
- "ISSUE ERROR in response to ISSUE PREPARE" on page 129
- "ISSUE ABEND in response to SYNCPOINT" on page 130
- "ISSUE ABEND in response to ISSUE PREPARE" on page 131
- "Session failure in response to SYNCPOINT" on page 132
- "Session failure in response to ISSUE PREPARE" on page 134
- "Session failure in response to SYNCPOINT ROLLBACK" on page 135.

SYNCPOINT in response to SYNCPOINT

Figure 26, Figure 27, and Figure 28 on page 124 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.

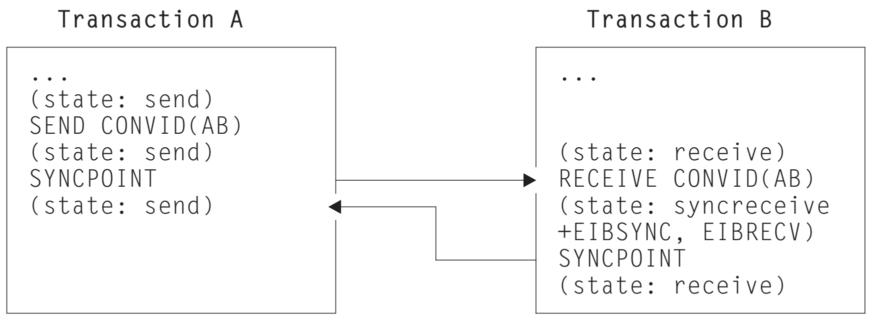

Figure 26. SYNCPOINT (in response to SEND followed by SYNCPOINT) on an APPC mapped conversation.

In this figure, transaction A is communicating with transaction B using an APPC mapped conversation. Initially, transaction A's conversation with B is in **send state***, and transaction B's conversation with A is in* **receive state***.*

1. *Transaction A issues a SEND command; its conversation remains in* **send state***.*
2. *Transaction B issues a RECEIVE command; it is suspended until data is received from transaction A.*
3. *Transaction A issues a SYNCPOINT command; outstanding data, and the syncpoint request, are transmitted. The transaction is suspended until the syncpoint response is received from transaction B.*
4. *Transaction B's RECEIVE command completes; EIBSYNC and EIBRECV are set, and its conversation is in* **syncreceive state***.*
5. *Transaction B issues a SYNCPOINT command; the response is transmitted, and its conversation is in* **receive state***.*
6. *Transaction A's SYNCPOINT command completes, and its conversation is in* **send state***.*

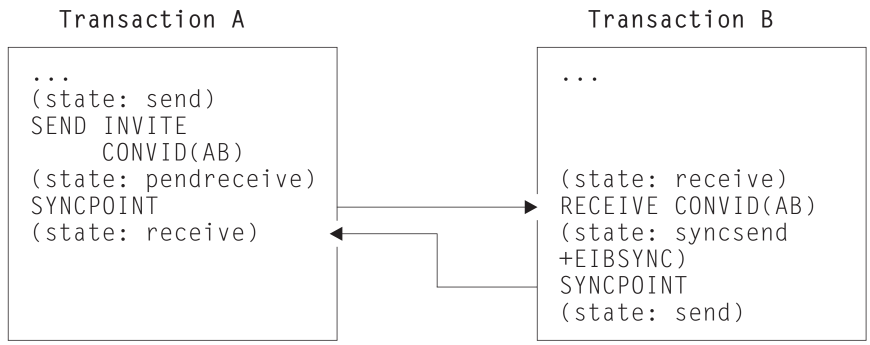

Figure 27. SYNCPOINT (in response to SEND INVITE followed by SYNCPOINT) on an APPC mapped conversation.

In this figure, transaction A is communicating with transaction B using an APPC mapped conversation. Initially, transaction A's conversation with B is in **send state***, and transaction B's conversation with A is in* **receive state***.*

1. *Transaction A issues a SEND INVITE command; its conversation is now in* **pendreceive state***.*
2. *Transaction B issues a RECEIVE command; it is suspended until data is received from transaction A.*
3. *Transaction A issues a SYNCPOINT command; outstanding data, the INVITE flag, and the syncpoint request, are transmitted. The transaction is suspended until the syncpoint response is received from transaction B.*
4. *Transaction B's RECEIVE command completes; EIBSYNC is set, and its conversation is in* **syncsend state***.*
5. *Transaction B issues a SYNCPOINT command; the response is transmitted, and its conversation is in* **send state***.*
6. *Transaction A's SYNCPOINT command completes, and its conversation is in* **receive state***.*

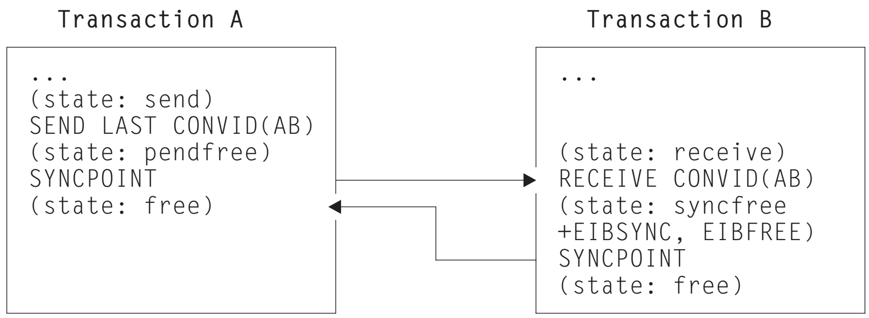

Figure 28. SYNCPOINT (in response to SEND LAST followed by SYNCPOINT) on an APPC mapped conversation.

In this figure, transaction A is communicating with transaction B using an APPC mapped conversation. Initially, transaction A's conversation with B is in **send state***, and transaction B's conversation with A is in* **receive state***.*

1. *Transaction A issues a SEND LAST command; its conversation is now in* **pendfree state***.*
2. *Transaction B issues a RECEIVE command; it is suspended until data is received from transaction A.*
3. *Transaction A issues a SYNCPOINT command; outstanding data, the LAST flag, and the syncpoint request, are transmitted. The transaction is suspended until the syncpoint response is received from transaction B.*
4. *Transaction B's RECEIVE command completes; EIBSYNC and EIBFREE are set, and its conversation is in* **syncfree state***.*
5. *Transaction B issues a SYNCPOINT command; the response is transmitted, and its conversation is in* **free state***.*
6. *Transaction A's SYNCPOINT command completes, and its conversation is in* **free state***.*

SYNCPOINT in response to ISSUE PREPARE

Figure 29 on page 125 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 29 on page 125 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 ATCV.

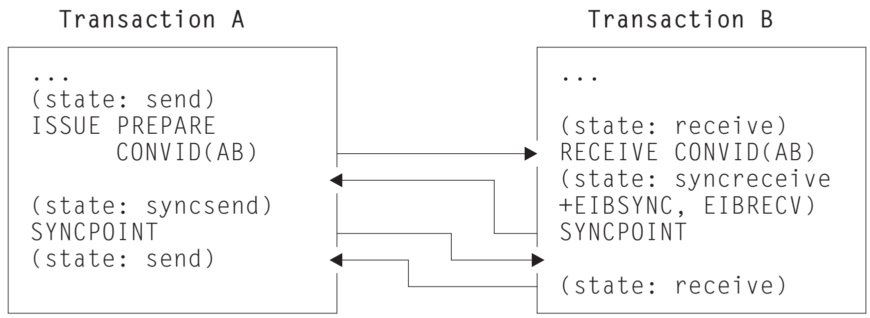

Figure 29. SYNCPOINT in response to ISSUE PREPARE on an APPC mapped conversation.

In this figure, transaction A is communicating with transaction B using an APPC mapped conversation. Initially, transaction A's conversation with B is in **send state***, and transaction B's conversation with A is in* **receive state***.*

1. *Transaction A issues an ISSUE PREPARE command; the prepare request is transmitted, and the transaction is suspended until a syncpoint request is received from transaction B*
2. *Transaction B issues a RECEIVE command which returns control immediately; EIBSYNC and EIBRECV are set, and its conversation is in* **syncreceive state***.*
3. *Transaction B issues a SYNCPOINT command; the syncpoint request is transmitted, and the transaction is suspended until the syncpoint response is received from transaction A.*
4. *Transaction A's ISSUE PREPARE command completes; its conversation is in* **syncsend state***.*
5. *Transaction A issues a SYNCPOINT command; the response is transmitted, and its conversation is in* **send state***.*
6. *Transaction B's SYNCPOINT command completes, and its conversation is in* **receive state***.*

SYNCPOINT ROLLBACK in response to SYNCPOINT ROLLBACK

Figure 30 on page 126 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.

```
Transaction A                           Transaction B

...                                     ...
(state: send)                           (state: receive)
SYNCPOINT ROLLBACK      ------------->  RECEIVE CONVID(AB)
                        <-------+       (state: rollback
(state: same as when            |       +EIBERR, EIBSYNRB)
unit of work began)             |
                                +-----  SYNCPOINT ROLLBACK
                                        (state: same as when
                                        unit of work began)
```

Figure 30. SYNCPOINT ROLLBACK in response to SYNCPOINT ROLLBACK on an APPC mapped conversation.

In this figure, transaction A is communicating with transaction B using an APPC mapped conversation. Initially, transaction A's conversation with B is in **send state***, and transaction B's conversation with A is in* **receive state***.*

1. *Transaction A issues a SYNCPOINT ROLLBACK command; the rollback request is transmitted, and the transaction is suspended until a rollback response is received from transaction B*
2. *Transaction B issues a RECEIVE command which returns control immediately; EIBERR and EIBSYNRB are set, and its conversation is in* **rollback state***.*
3. *Transaction B issues a SYNCPOINT ROLLBACK command; the rollback response is transmitted; transaction B's conversation is restored to the state it was in at the start of the unit of work.*
4. *Transaction A's SYNCPOINT ROLLBACK command completes; its conversation is restored to the state it was in at the start of the unit of work.*

SYNCPOINT ROLLBACK in response to SYNCPOINT

Figure 31 on page 127 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.

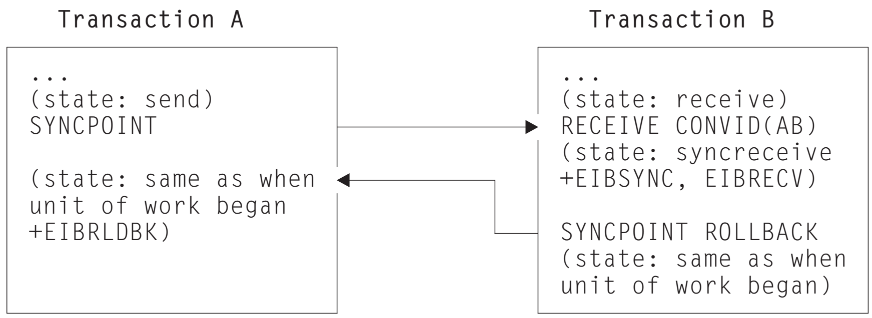

Figure 31. SYNCPOINT ROLLBACK in response to SYNCPOINT on an APPC mapped conversation. In this figure, transaction A is communicating with transaction B using an APPC mapped conversation. Initially, transaction A's conversation with B is in **send state***, and transaction B's conversation with A is in* **receive state***.*

1. *Transaction A issues a SYNCPOINT command; the syncpoint request is transmitted, and the transaction is suspended until a response is received from transaction B.*
2. *Transaction B issues a RECEIVE command which returns control immediately; EIBSYNC and EIBRECV are set, and its conversation is in* **syncreceive state***.*
3. *Transaction B issues a SYNCPOINT ROLLBACK command; the rollback response is transmitted; transaction B's conversation is restored to the state it was in at the start of the unit of work.*
4. *Transaction A's SYNCPOINT command completes; EIBRLDBK is set; transaction A's conversation is restored to the state it was in at the start of the unit of work.*

SYNCPOINT ROLLBACK in response to ISSUE PREPARE

Figure 32 on page 128 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.

```
Transaction A                     Transaction B

...                               ...
(state: send)                     (state: receive)
ISSUE PREPARE                     RECEIVE CONVID(AB)
      CONVID(AB)                  (state: syncreceive
(state: rollback                  +EIBSYNC, EIBRECV)
+EIBERR, EIBSYNRB)
SYNCPOINT ROLLBACK                SYNCPOINT ROLLBACK
(state: same as when              (state: same as when
unit of work began)               unit of work began)
```

Figure 32. SYNCPOINT ROLLBACK in response to ISSUE PREPARE on an APPC mapped conversation.

In this figure, transaction A is communicating with transaction B using an APPC mapped conversation. Initially, transaction A's conversation with B is in **send state***, and transaction B's conversation with A is in* **receive state***.*

1. *Transaction A issues an ISSUE PREPARE command; the syncpoint request is transmitted, and the transaction is suspended until a response is received from transaction B.*
2. *Transaction B issues a RECEIVE command which returns control immediately; EIBSYNC and EIBRECV are set, and its conversation is in* **syncreceive state***.*
3. *Transaction B issues a SYNCPOINT ROLLBACK command; the rollback response is transmitted; transaction B's conversation is restored to the state it was in at the start of the unit of work.*
4. *Transaction A's ISSUE PREPARE command completes; EIBERR and EIBRLDBK are set; transaction A's conversation is restored to the state it was in at the start of the unit of work.*

ISSUE ERROR in response to SYNCPOINT

Figure 33 on page 129 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.

Note that if transaction A were running on a CICS release earlier than 3.2, the results would be different. (See the *Intercommunication Guide* for the relevant release.)

```
   Transaction A                     Transaction B

 ...                               ...
 (state: send)                     (state: receive)
 SYNCPOINT          ------------>  RECEIVE CONVID(AB)
                                   (state: syncreceive
                                   +EIBSYNC, EIBRECV)
                                   ISSUE ERROR
                                         CONVID (AB)
                                   (state: send)
                    <------------  SEND INVITE WAIT
                                        CONVID(AB)
                                   (state: receive)
                    ------------>  RECEIVE CONVID(AB)
                                   (state: rollback
                                   +EIBERR, EIBSYNRB)
 (state: same as when <----------  SYNCPOINT ROLLBACK
 unit of work began                (state: same as when
 +EIBRLDBK)                        unit of work began)
```

Figure 33. ISSUE ERROR in response to SYNCPOINT on an APPC mapped conversation.

In this figure, transaction A is communicating with transaction B using an APPC mapped conversation. Initially, transaction A's conversation with B is in **send state***, and transaction B's conversation with A is in* **receive state***.*

1. *Transaction A issues a SYNCPOINT command; the syncpoint request is transmitted, and the transaction is suspended until a response is received from transaction B.*
2. *Transaction B issues a RECEIVE command which returns control immediately; EIBSYNC and EIBRECV are set, and its conversation is in* **syncreceive state***.*
3. *Transaction B issues an ISSUE ERROR command; its conversation is in* **send state***.*
4. *Transaction B issues a SEND INVITE WAIT command; the error indication and the INVITE flag are transmitted to transaction A; transaction B's conversation is in* **receive state***.*
5. *On behalf of transaction A, CICS sends a rollback request to transaction B.*
6. *Transaction B issues a RECEIVE command which returns control immediately; EIBERR and EIBSYNRB are set, and its conversation is in* **rollback state***.*
7. *Transaction B issues a SYNCPOINT ROLLBACK command; the rollback response is transmitted; transaction B's conversation is restored to the state it was in at the start of the unit of work.*
8. *Transaction A's SYNCPOINT command completes; EIBRLDBK is set; transaction A's conversation is restored to the state it was in at the start of the unit of work.*

ISSUE ERROR in response to ISSUE PREPARE

Figure 34 on page 130 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.

```
   Transaction A                          Transaction B

 ...                                    ...
 (state: send)                          (state: receive)
 ISSUE PREPARE          ------------->  RECEIVE CONVID(AB)
       CONVID(AB)                       (state: syncreceive
                                        +EIBSYNC, EIBRECV)
                                        ISSUE ERROR
                                              CONVID (AB)
                                        (state: send)
 (state: receive        <-------------  WAIT CONVID(AB)
 +EIBERR)                               (state: send)
```

Figure 34. ISSUE ERROR in response to ISSUE PREPARE on an APPC mapped conversation.

In this figure, transaction A is communicating with transaction B using an APPC mapped conversation. Initially, transaction A's conversation with B is in **send state***, and transaction B's conversation with A is in* **receive state***.*

1. *Transaction A issues an ISSUE PREPARE command; the syncpoint request is transmitted, and the transaction is suspended until a response is received from transaction B.*
2. *Transaction B issues a RECEIVE command which returns control immediately; EIBSYNC and EIBRECV are set, and its conversation is in* **syncreceive state***.*
3. *Transaction B issues an ISSUE ERROR command; its conversation is in* **send state***.*
4. *Transaction B issues a WAIT command; the error indication is transmitted to transaction A; transaction B's conversation is in* **send state***.*
5. *Transaction A's ISSUE PREPARE command completes; EIBERR is set; transaction A's conversation is in* **receive state***.*

ISSUE ABEND in response to SYNCPOINT

Figure 35 on page 131 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.

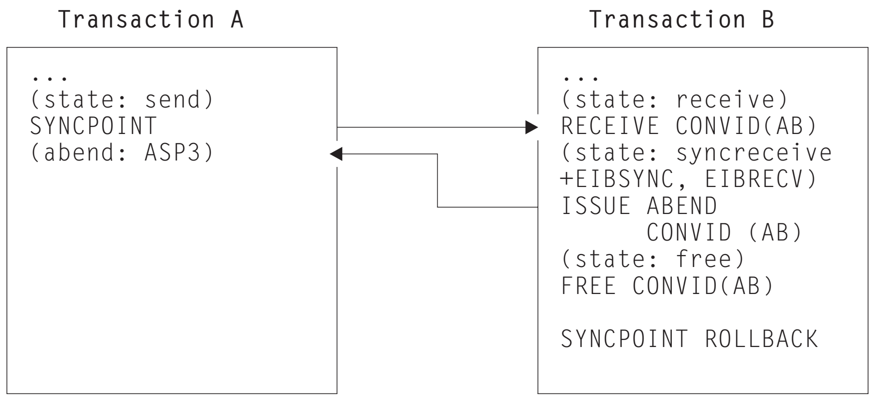

Figure 35. ISSUE ABEND in response to SYNCPOINT on an APPC mapped conversation.

In this figure, transaction A is communicating with transaction B using an APPC mapped conversation. Initially, transaction A's conversation with B is in **send state***, and transaction B's conversation with A is in* **receive state***.*

1. *Transaction A issues a SYNCPOINT command; the syncpoint request is transmitted, and the transaction is suspended until a response is received from transaction B.*
2. *Transaction B issues a RECEIVE command which returns control immediately; EIBSYNC and EIBRECV are set, and its conversation is in* **syncreceive state***.*
3. *Transaction B issues an ISSUE ABEND command; theabend indication is transmitted; transaction B's conversation is in* **free state***.*
4. *Transaction A abends with code ASP3.*
5. *Transaction B issues a FREE command to free its conversation with transaction A.*
6. *Transaction B issues a SYNCPOINT ROLLBACK command in order that its resources remain consistent with transaction A's resources.*

ISSUE ABEND in response to ISSUE PREPARE

Figure 36 on page 132 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.

Transaction A	Transaction B

...
(state: send)
ISSUE PREPARE
      CONVID(AB)
(state: free
+ EIBERR, EIBFREE)
FREE CONVID(AB)
 	
SYNCPOINT ROLLBACK
	...
	(state: receive)
	RECEIVE CONVID(AB)
	(state: syncreceive
	+EIBSYNC, EIBRECV)
	ISSUE ABEND
	      CONVID(AB)
	(state: free)
	FREE CONVID(AB)
	
	SYNCPOINT ROLLBACK

Figure 36. ISSUE ABEND in response to ISSUE PREPARE on an APPC mapped conversation.

In this figure, transaction A is communicating with transaction B using an APPC mapped conversation. Initially, transaction A's conversation with B is in **send state***, and transaction B's conversation with A is in* **receive state***.*

1. *Transaction A issues an ISSUE PREPARE command; the syncpoint request is transmitted, and the transaction is suspended until a response is received from transaction B.*
2. *Transaction B issues a RECEIVE command which returns control immediately; EIBSYNC and EIBRECV are set, and its conversation is in* **syncreceive state***.*
3. *Transaction B issues an ISSUE ABEND command; theabend indication is transmitted; transaction B's conversation is in* **free state***.*
4. *Transaction A's ISSUE PREPARE command completes; EIBERR and EIBFREE are set; transaction A's conversation is in* **free state***.*

From this point, the two transactions are independent of one another.

5. *Transaction A issues a FREE command to free its conversation with transaction A.*
6. *Transaction A issues a SYNCPOINT ROLLBACK command in order that its resources remain consistent with transaction A's resources.*
7. *Transaction B issues a FREE command to free its conversation with transaction A.*
8. *Transaction B issues a SYNCPOINT ROLLBACK command in order that its resources remain consistent with transaction A's resources.*

Session failure in response to SYNCPOINT

Figure 37 on page 133 and Figure 38 on page 134 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.

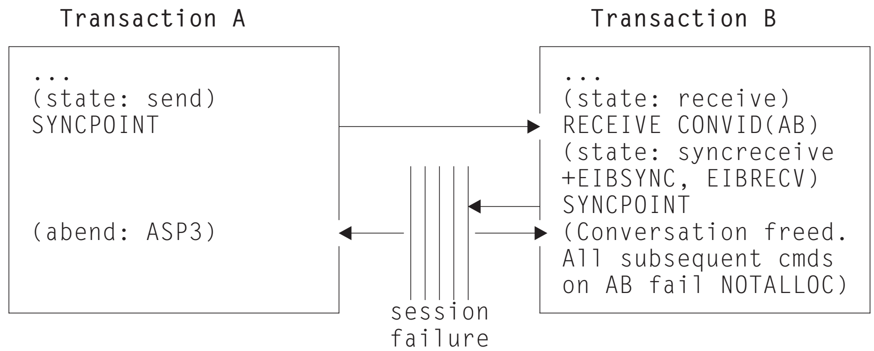

Figure 37. Session failure before SYNCPOINT in response to SYNCPOINT on an APPC mapped conversation.

In this figure, transaction A is communicating with transaction B using an APPC mapped conversation. Initially, transaction A's conversation with B is in **send state***, and transaction B's conversation with A is in* **receive state***.*

1. *Transaction A issues a SYNCPOINT command; the syncpoint request is transmitted, and the transaction is suspended until a response is received from transaction B.*
2. *Transaction B issues a RECEIVE command which returns control immediately; EIBSYNC and EIBRECV are set, and its conversation is in* **syncreceive state***.*
3. *The session between transaction A and transaction B fails.*

From this point , the two transactions are independent of one another.

4. *Transaction B issues a SYNCPOINT command; its conversation is freed, and a NOTALLOC condiition is raised for any further commands that attempt to use the conversation.*
5. *Transaction A abends with code ASP3.*

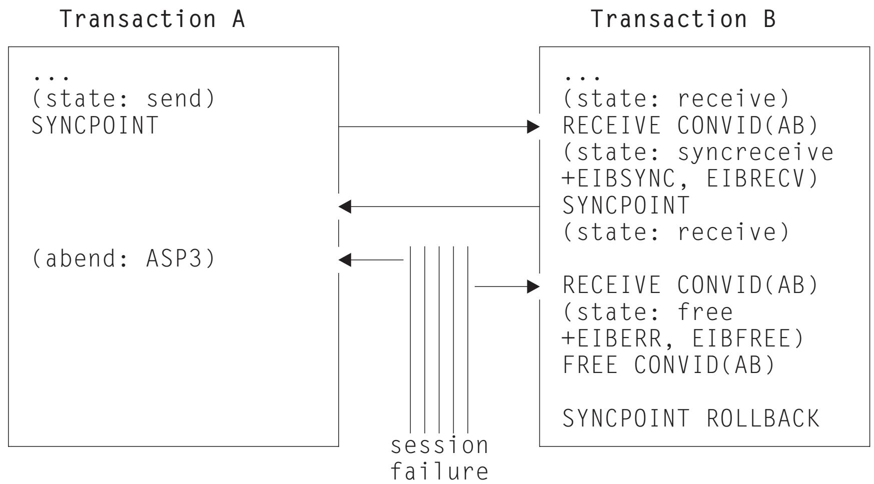

Figure 38. Session failure after SYNCPOINT in response to SYNCPOINT on an APPC mapped conversation.

In this figure, transaction A is communicating with transaction B using an APPC mapped conversation. Initially, transaction A's conversation with B is in **send state***, and transaction B's conversation with A is in* **receive state***.*

1. *Transaction A issues a SYNCPOINT command; the syncpoint request is transmitted, and the transaction is suspended until a response is received from transaction B.*
2. *Transaction B issues a RECEIVE command which returns control immediately; EIBSYNC and EIBRECV are set, and its conversation is in* **syncreceive state***.*
3. *Transaction B issues a SYNCPOINT command; the response is transmitted, and its conversation is in* **receive state***.*
4. *The session between transaction A and transaction B fails.*

From this point , the two transactions are independent of one another.

5. *Transaction A abends with code ASP3.*
6. *Transaction B issues a RECEIVE command which returns control immediately; EIBERR and EIBFREE are set, and its conversation is in* **free state***.*
7. *Transaction B issues a FREE command to free its conversation with transaction A.*
8. *Transaction B issues a SYNCPOINT ROLLBACK command in order that its resources remain consistent with transaction A's resources.*

Session failure in response to ISSUE PREPARE

Figure 39 on page 135 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.

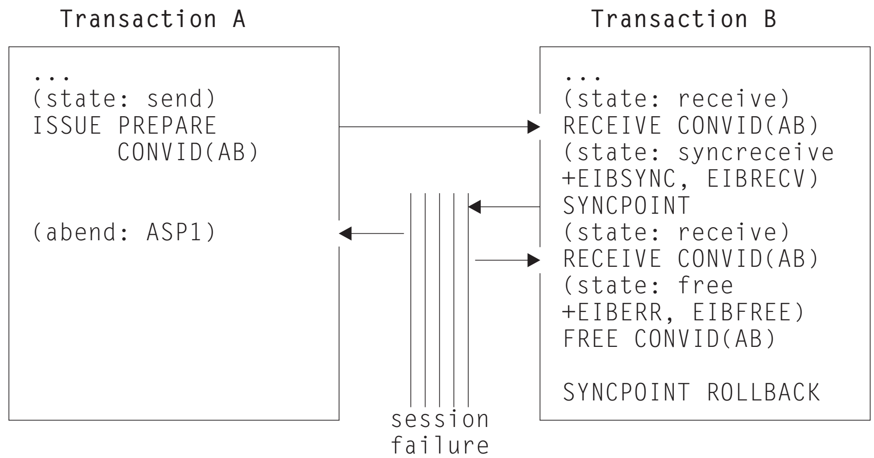

Figure 39. Session failure during SYNCPOINT in response to ISSUE PREPARE on an APPC mapped conversation.

In this figure, transaction A is communicating with transaction B using an APPC mapped conversation. Initially, transaction A's conversation with B is in **send state***, and transaction B's conversation with A is in* **receive state***.*

1. *Transaction A issues an ISSUE PREPARE command; the syncpoint request is transmitted, and the transaction is suspended until a response is received from transaction B.*
2. *Transaction B issues a RECEIVE command which returns control immediately; EIBSYNC and EIBRECV are set, and its conversation is in* **syncreceive state***.*
3. *The session between transaction A and transaction B fails.*

From this point , the two transactions are independent of one another.

4. *Transaction A abends with code ASP1.*
5. *Transaction B issues a SYNCPOINT command; its conversation is in* **receive state***.*
6. *Transaction B issues a RECEIVE command which returns control immediately; EIBERR and EIBFREE are set, and its conversation is in* **free state***.*
7. *Transaction B issues a FREE command to free its conversation with transaction A.*
8. *Transaction B issues a SYNCPOINT ROLLBACK command in order that its resources remain consistent with transaction A's resources.*

Session failure in response to SYNCPOINT ROLLBACK

Figure 40 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.

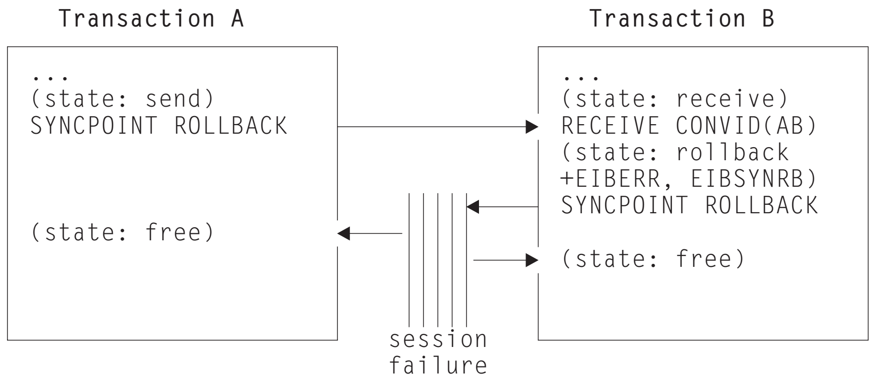

Figure 40. Session failure during SYNCPOINT ROLLBACK in response to SYNCPOINT ROLLBACK on an APPC mapped conversation.

In this figure, transaction A is communicating with transaction B using an APPC mapped conversation. Initially, transaction A's conversation with B is in ***send state****, and transaction B's conversation with A is in* ***receive state****.*

1. *Transaction A issues a SYNCPOINT ROLLBACK command; the rollback request is transmitted, and the transaction is suspended until a rollback response is received from transaction B*
2. *Transaction B issues a RECEIVE command which returns control immediately; EIBERR and EIBSYNRB are set, and its conversation is in* ***rollback state****.*
3. *The session between transaction A and transaction B fails.*

From this point , the two transactions are independent of one another.

4. *Transaction B issues a SYNCPOINT ROLLBACK command; its conversation is in* ***free state****.*
5. *Transaction A's SYNCPOINT ROLLBACK command completes; its conversation is in* ***free state****.*

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.

SYNCPOINT in response to SYNCPOINT

Figure 41 on page 138 shows the sequence of events for a successful syncpoint involving six conversing transactions:

Transaction A

- is in conversation with transactions B and D. Before the syncpoint, its conversations with B and D are in send state.
- is the syncpoint initiator with respect to transactions B and D.

Transaction B

- is in conversation with transactions A, C, and E. Before the syncpoint, its conversation with A is in receive state, and its conversations with C and E are in send state.
- is a syncpoint agent of transaction A, and the syncpoint initiator with respect to transactions C and E.

Transaction C

- is in conversation with transaction B. Before the syncpoint, its conversation with B is in receive state.
- is a syncpoint agent of transaction B.

Transaction D

- is in conversation with transactions A and F. Before the syncpoint, its conversation with A is in receive state, and its conversation F is in send state.
- is a syncpoint agent of transaction A, and the syncpoint initiator with respect to transaction F.

Transaction E

- is in conversation with transaction B. Before the syncpoint, its conversation with B is in receive state.
- is a syncpoint agent with respect to transaction B.

Transaction F

- is in conversation with transaction D. Before the syncpoint, its conversation with D is in receive state.
- is the only syncpoint agent of transaction D.

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).

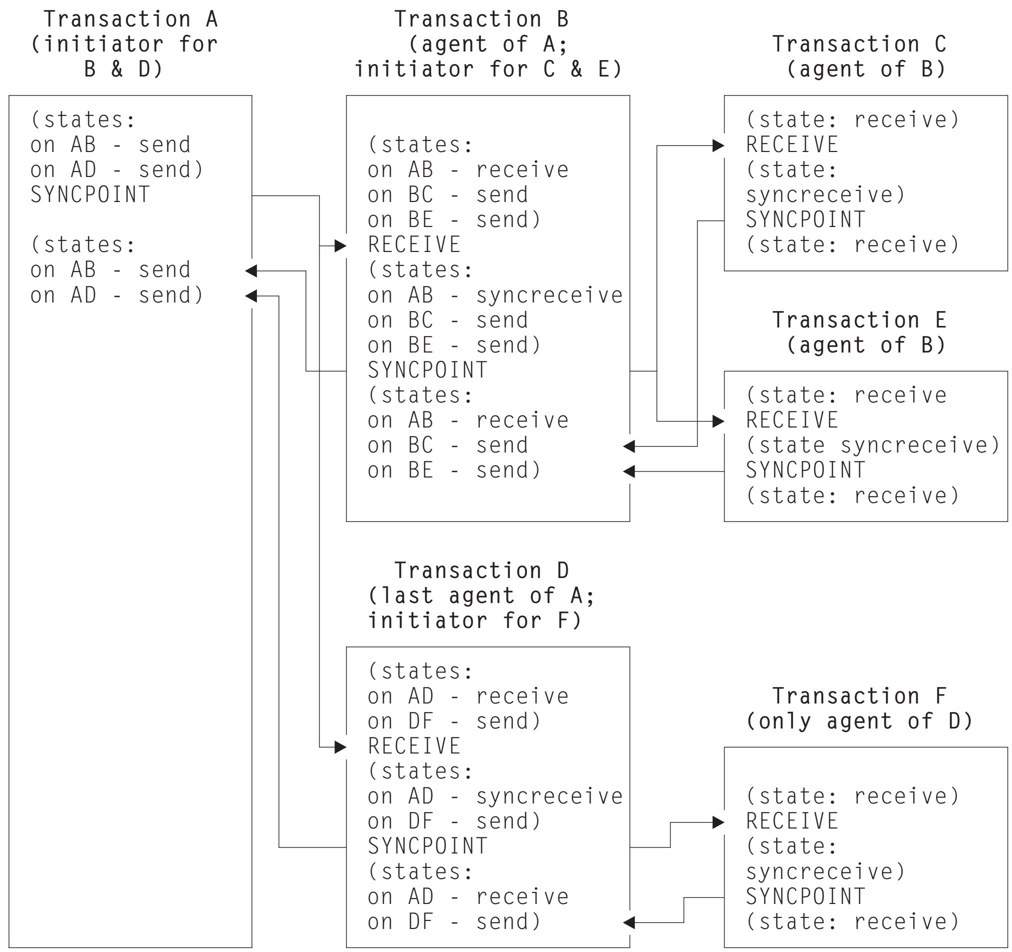

Figure 41. A distributed syncpoint with all partners running on CICS Transaction Server for z/OS, Version 2 Release 2

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.

SYNCPOINT ROLLBACK in response to SYNCPOINT

Figure 42 on page 140 shows the sequence of events for a syncpoint involving six conversing transactions, when one of the agents determines that the distributed transaction should be backed out. The topology, and initial states are the same as in Figure 41 on page 138:

Transaction A
- is in conversation with transactions B and D. Before the syncpoint, its conversations with B and D are in send state.
- is the syncpoint initiator with respect to transactions B and D.

Transaction B
- is in conversation with transactions A, C, and E. Before the syncpoint, its conversation with A is in receive state, and its conversations with C and E are in send state.
- is a syncpoint agent of transaction A, and the syncpoint initiator with respect to transactions C and E.

Transaction C
- is in conversation with transaction B. Before the syncpoint, its conversation with B is in receive state.
- is a syncpoint agent of transaction B.

Transaction D
- is in conversation with transactions A and F. Before the syncpoint, its conversation with A is in receive state, and its conversation F is in send state.

- is a syncpoint agent of transaction A, and the syncpoint initiator with respect to transaction F.

Transaction E

- is in conversation with transaction B. Before the syncpoint, its conversation with B is in receive state.
- is a syncpoint agent with respect to transaction B.

Transaction F

- is in conversation with transaction D. Before the syncpoint, its conversation with D is in receive state.
- is the only syncpoint agent of transaction D.

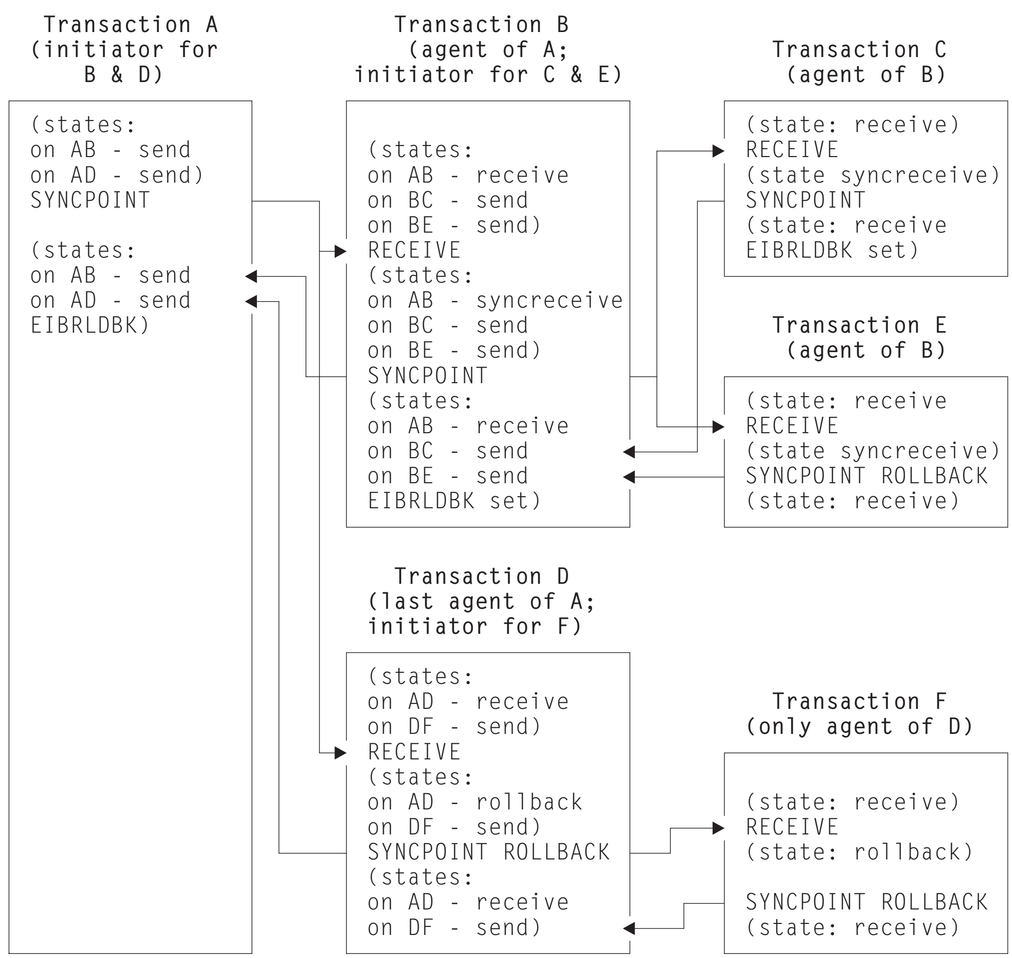

Figure 42. Rollback during distributed syncpointing

As in Figure 41 on page 138, 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 indoubt 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 **indoubt 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 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 43. 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 UOW is complete.

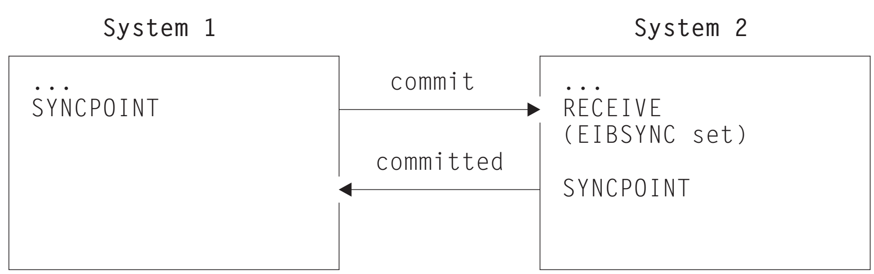

Figure 43. Syncpoint flows in a single conversation

When the syncpoint agent has a conversation with a third transaction, Figure 44 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.

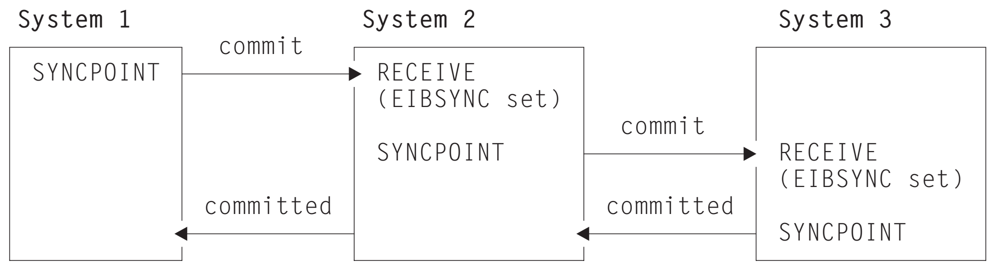

Figure 44. Syncpoint flows in concurrent conversations

When the syncpoint initiator has two concurrent conversations, the flows involved are shown in Figure 45. 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.

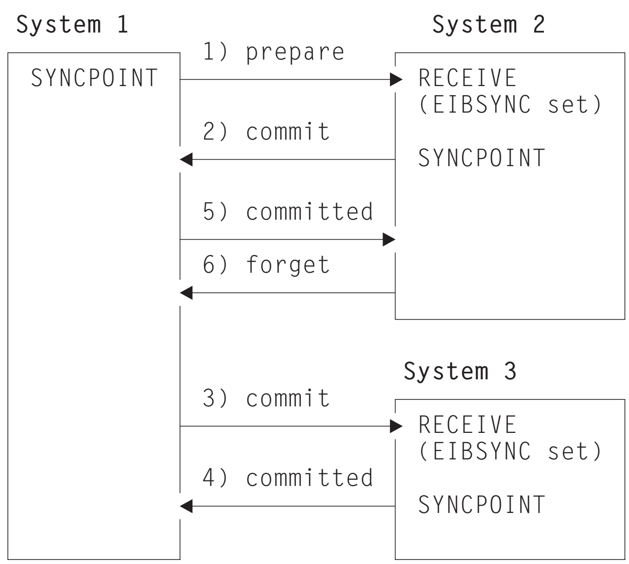

Figure 45. Syncpoint flows in concurrent conversations with one initiator. The initiator uses only SYNCPOINT.

If the syncpoint initiator decides to prepare the conversation with system 2 explicitly before issuing a syncpoint, the flows involved are shown in Figure 46 on page 143. In this case, the application program in system 1 issues an ISSUE PREPARE command, followed by SYNCPOINT command, rather than just a SYNCPOINT command; however, 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.

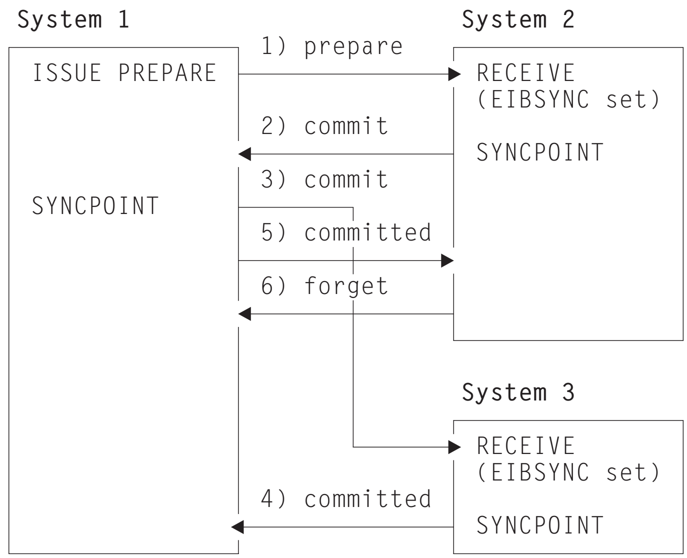

Figure 46. Syncpoint flows in concurrent conversations with one initiator. The initiator uses ISSUE PREPARE before SYNCPOINT.

For further information on the flows in a distributed process, see the *SNA Reference: Peer Protocols* book.

Part 7. Appendixes

Appendix A. CICS mapping to the APPC architecture

This appendix shows how the APPC programming language (described in *Transaction Programmer's Reference Manual for LU Type 6.2*, GC30-3084) is implemented by CICS.

The appendix contains three main sections:

1. "Command mapping for APPC basic conversations" on page 148

   The CICS application programming interface for basic, or unmapped, conversations is described in "Chapter 8. State transitions in APPC basic conversations" on page 93. These tables show how the APPC verbs map to the EXEC CICS commands.
2. "Command mapping for APPC mapped conversations" on page 155

   The CICS application programming interface for mapped conversations is described in "Chapter 4. State transitions in APPC mapped conversations" on page 41. For programming information about the full syntax of EXEC CICS commands for APPC mapped conversations, see the *CICS Application Programming Reference* manual. These tables show how the APPC verbs map to the EXEC CICS commands.
3. "CICS deviations from the APPC architecture" on page 161

   How the CICS APIs differ from the APPC architecture and their effects on the CICS application programmer are discussed.

For information on which APPC option sets are supported by CICS and which are not, or on how CICS implements the APPC control operator verbs, see the *CICS Intercommunication Guide*.

Command mapping for APPC basic conversations

The following tables show the mapping between APPC verbs and CICS commands for basic conversations. See "Return codes for APPC basic conversations" on page 153 for details of the corresponding return code mapping.

ALLOCATE	EXEC CICS GDS ALLOCATE + EXEC CICS GDS CONNECT PROCESS
LU_NAME(vble)	SYSID on ALLOCATE
MODE_NAME(vble)	MODENAME on ALLOCATE
MODE_NAME('SNASVCMG')	MODENAME on ALLOCATE
TPN(vble)	PROCNAME on CONNECT PROCESS (with PROCLENGTH)
TYPE(BASIC_CONVERSATION)	Supported by GDS
TYPE(MAPPED_CONVERSATION)	Not supported
RETURN_CONTROL(WHEN_SESSION_ALLOCATED)	Default on ALLOCATE
RETURN_CONTROL(WHEN_CONWINNER_ALLOCATED)	Not supported
RETURN_CONTROL (WHEN_CONVERSATION_GROUP_ALLOCATED)	Supported
RETURN_CONTROL(IMMEDIATE)	NOQUEUE/NOSUSPEND on ALLOCATE
SYNC_LEVEL	SYNCLEVEL on CONNECT PROCESS 0 — None 1 — Confirm 2 — Syncpoint
SECURITY(NONE)	Not supported
SECURITY(SAME)	Default on ALLOCATE
SECURITY(PGM(USED_ID(vble)	Not supported
(PASSWORD(vble)))	Not supported
PIP(NO)	Supported by PIPLENGTH(0)
PIP(YES(vble1,vble2 ... vblen))	Supported by PIPLIST+PIPLENGTH
RESOURCE	Returned by GDS ASSIGN
RETURN_CODE	Supported

BACKOUT	EXEC CICS SYNCPOINT ROLLBACK
RETURN_CODE	Supported

CONFIRM	EXEC CICS GDS CONFIRM
RESOURCE	CONVID
RETURN_CODE	Supported
REQUEST_TO_SEND_RECEIVED	Returned in CDBSIG

CONFIRMED	EXEC CICS GDS ISSUE CONFIRMATION
RESOURCE	CONVID
RETURN_CODE	Supported

DEALLOCATE	**EXEC CICS GDS SEND LAST** + **EXEC CICS SYNCPOINT** + **EXEC CICS GDS FREE**
TYPE(SYNC_LEVEL) None	EXEC CICS GDS SEND LAST WAIT + EXEC CICS GDS FREE
TYPE(SYNC_LEVEL) Confirm	EXEC CICS GDS SEND LAST CONFIRM + EXEC CICS GDS FREE
TYPE(SYNC_LEVEL) Syncpt	EXEC CICS GDS SEND LAST + EXEC CICS SYNCPOINT + EXEC CICS GDS FREE
TYPE(FLUSH)	EXEC CICS GDS SEND LAST WAIT + EXEC CICS GDS FREE
TYPE(CONFIRM)	EXEC CICS GDS SEND LAST CONFIRM + EXEC CICS GDS FREE
TYPE(ABEND_PROG) Depends on setting of CDBFREE by previous command:	
CDBFREE = X'00	EXEC CICS GDS ISSUE ABEND + EXEC CICS GDS FREE
CDBFREE = X'FF	EXEC CICS GDS FREE
TYPE(ABEND_SVC)	Not supported at API (option set 11)
TYPE(ABEND_TIMER)	Not supported at API (option set 11)
TYPE(LOCAL)	EXEC CICS GDS FREE
LOG_DATA(vble)	Not available at API. CICS inserts the appropriate values
RETURN_CODE	Supported

FLUSH	**EXEC CICS GDS WAIT**

GET_ATTRIBUTES	**EXEC CICS GDS EXTRACT PROCESS** or **EXEC CICS GDS ASSIGN** or **EXEC CICS ASSIGN**
RESOURCE	CONVID
SYNC_LEVEL	SYNCLEVEL on GDS EXTRACT PROCESS 0 — None 1 — Confirm 2 — Syncpoint
UOW_IDENTIFIER	See note
OWN_FULLY_QUALIFIED_LU_NAME	See note
PARTNER_LU_NAME	GDS ASSIGN PRINSYSID
PARTNER_FULLY_QUALIFIED_LU_NAME	See note
MODE_NAME	See note
USERID	ASSIGN USERID
	Note: These values are not normally required in CICS applications and are not available at the API.
RETURN_CODE	Supported

GET_TYPE	**EXEC CICS GDS ASSIGN** (+ return code test)
RESOURCE	PRINCONVID
TYPE(vble)	RETCODE clear = GDS (BASIC) 03 04 = wrong conversation level

POST_ON_RECEIPT	**Not supported**

PREPARE_FOR_SYNCPT	**EXEC CICS GDS ISSUE PREPARE**
RESOURCE	CONVID
RETURN_CODE	Supported

PREPARE_TO_RECEIVE	**EXEC CICS GDS SEND INVITE**
TYPE(SYNC_LEVEL) none	EXEC CICS GDS SEND INVITE WAIT
TYPE(SYNC_LEVEL) confirm	EXEC CICS GDS SEND INVITE CONFIRM
TYPE(SYNC_LEVEL) syncpt	EXEC CICS GDS SEND INVITE + EXEC CICS SYNCPOINT
TYPE(FLUSH)	EXEC CICS GDS SEND INVITE WAIT
TYPE(CONFIRM)	EXEC CICS GDS SEND INVITE CONFIRM
LOCKS(SHORT)	Defaulted
LOCKS(LONG)	Not supported
RETURN_CODE	Supported

RECEIVE_AND_WAIT	**EXEC CICS GDS RECEIVE** (for both LL and BUFFER)
RESOURCE	CONVID field
FILL(BUFFER)	BUFFER option
FILL(LL)	LLID option
LENGTH(vble) Input	MAXFLENGTH option
LENGTH(vble) Output	FLENGTH option
RETURN_CODE	Supported
REQUEST_TO_SEND_RECEIVED	Returned in CDBSIG
DATA	INTO or SET option
WHAT_RECEIVED CONFIRM CONFIRM_DEALLOCATE CONFIRM_SEND DATA DATA_COMPLETE DATA_INCOMPLETE LL_TRUNCATED SEND TAKE_SYNCPT TAKE_SYNCPT_DEALLOCATE TAKE_SYNCPT_SEND	CICS Settings CDBCONF + CDBRECV CDBCONF + CDBFREE CDBCONF FLENGTH field ¬= 0 [+ CDBRECV] CDBCOMPL [+ CDBRECV] ¬CDBCOMPL [+ CDBRECV] RETCODE = X'0310....' ¬CDBRECV CDBSYNC + CDBRECV CDBSYNC + CDBFREE CDBSYNC

Notes:

1. Mapping of RECEIVE_AND_WAIT to EXEC CICS GDS RECEIVE is not always one to one.

   When a CICS RECEIVE command is issued, CICS returns all the information and data (the DATA, the WHAT_RECEIVED flags, and the RETURN_CODE) at once. On completion of a CICS command, more than one indicator may be set, as shown in the WHAT_RECEIVED mapping above. It may be necessary to perform more than one subsequent command to honor the actions required by the indicators. For this reason, the action flags must be saved when they are received, and then acted on one by one. If the same data area is used for CONVDATA on successive GDS commands, the flags are overwritten and lost.

   APPC does not work this way; a RECEIVE_AND_WAIT verb returns either data or information about the conversation state (as indicated by WHAT_RECEIVED), but never both.

   It is necessary to program round this difference in philosophy when translating APPC verbs into CICS commands.
2. APPC allows a RECEIVE_AND_WAIT to be issued immediately after an ALLOCATE verb. When you are writing basic conversations in CICS, however, you must supply the PREPARE_TO_RECEIVE explicitly, as follows:

```
ALLOCATE                 EXEC CICS GDS ALLOCATE
                        +EXEC CICS CONNECT PROCESS
(Required by CICS)       EXEC CICS GDS SEND INVITE WAIT
RECEIVE_AND_WAIT         EXEC CICS GDS RECEIVE
```

REQUEST_TO_SEND	**EXEC CICS GDS ISSUE SIGNAL**
RESOURCE	CONVID field
RETURN_CODE	Supported

SEND_DATA	EXEC CICS GDS SEND
RESOURCE	CONVID field
DATA	FROM option
LENGTH	FLENGTH option
RETURN_CODE	Supported
REQUEST_TO_SEND_RECEIVED	Returned in CDBSIG
ENCRYPT	Not supported

SEND_ERROR	EXEC CICS GDS ISSUE ERROR
RESOURCE	CONVID field
TYPE(PROG)	Default
TYPE(SVC)	Not supported
LOG_DATA	Not supported
RETURN_CODE	Supported
REQUEST_TO_SEND_RECEIVED	Returned in CDBSIG

SYNCPT	EXEC CICS SYNCPOINT
RETURN_CODE	Zero - Control returned to program. Non-zero - CICS takes action; to backout the UOW (and abend the task or set EIBRLDBK).

Notes:

1. EXEC CICS SYNCPOINT is not a GDS command.
2. For certain specialized applications, the PREPARE flow (the first flow in syncpoint exchanges) may be sent for a particular conversation by using the command:

   ```
   EXEC CICS GDS ISSUE PREPARE
   ```

   This enables any outstanding messages in the network (for example, SEND ERROR) to be received before proceeding, or deciding not to proceed, with the full syncpoint.

TEST	Check CDB flags
RETURN_CODE	Not supported
TEST(POSTED)	Check CDB flags
TEST(REQUEST_TO_SEND_RECEIVED)	Check CDBSIG

WAIT	Not supported

Return codes for APPC basic conversations

APPC RETURN_CODE	CICS return codes
OK	CDBERR and RETCODE are zero
ALLOCATION_ERROR	
Local allocation failures:	CICS is unable to allocate a session for an ALLOCATE command.
ALLOCATION_FAILURE_NO_RETRY	RETCODE = 01.... The second and subsequent bytes give further information
ALLOCATION_FAILURE_RETRY	For temporary problems, CICS waits in the ALLOCATE command until the problem has cleared and then continues. See also the UNSUCCESSFUL return code, which relates to the NOQUEUE option on the CICS ALLOCATE command.
Remote allocation failures:	These are returned to the program after the CONNECT PROCESS command has been issued, and the partner system has been unable to start the requested task. They may be returned on any subsequent command that relates to the session in use
CONVERSATION_TYPE_MISMATCH	CDBERRCD = 10086034
PIP_NOT_ALLOWED	CDBERRCD = 10086031
PIP_NOT_SPECIFIED_CORRECTLY	CDBERRCD = 10086032
SECURITY_NOT_VALID	CDBERRCD = 080F6051
SYNC_LEVEL_NOT_SUPPORTED_BY_PGM	CDBERRCD = 10086041
SYNC_LEVEL_NOT_SUPPORTED_BY_LU	RETCODE = 030C Note: CICS remembers SYNC_LEVEL negotiated at bind time and does not permit a request to be sent for a sync level not supported by the remote LU.
TPN_NOT_RECOGNIZED	CDBERRCD = 10086021
TRANS_PGM_NOT_AVAIL_NO_RETRY	CDBERRCD = 084C0000
TRANS_PGM_NOT_AVAIL_RETRY	CDBERRCD = 084B6031
BACKED_OUT	CDBERRCD = 08240000
DEALLOCATE_ABEND_PROG	CDBERRCD = 08640000
DEALLOCATE_ABEND_SVC	CDBERRCD = 08640001
DEALLOCATE_ABEND_TIMER	CDBERRCD = 08640002
DEALLOCATE_NORMAL	CDBFREE + ¬CDBERR
PARAMETER_ERROR	RETCODE = 01 0C .. This return code relates ONLY to the ALLOCATE command (for CICS). It is given when an invalid LU name or MODE name has been specified. The third byte gives additional information.

APPC RETURN_CODE	CICS return codes
PROG_ERROR_NO_TRUNC PROG_ERROR_TRUNC PROG_ERROR_PURGING	CDBERRCD = 08890000 (RECEIVE Only) CDBERRCD = 08890001 CDBERRCD = 08890000
RESOURCE_FAILURE_RETRY RESOURCE_FAILURE_NO_RETRY	CDBERRCD = A000 CDBERRCD = A000
SVC_ERROR_NO_TRUNC SVC_ERROR_TRUNC SVC_ERROR_PURGING	CDBERRCD = 08890100 (RECEIVE Only) CDBERRCD = 08890101 CDBERRCD = 08890100
UNSUCCESSFUL This return code relates ONLY to the APPC ALLOCATE verb with RETURN_CONTROL(IMMEDIATE) specified. This is implemented in CICS with the NOQUEUE option on the ALLOCATE command.	RETCODE = 01 04 04 Control returned to the program because a session was not immediately available.
Note: In all cases, where a value for CDBERRCD is given, CDBERR will be set to X'FF'. It is intended that the program should first test CDBERR and then examine CDBERRCD if additional information is required.	

Command mapping for APPC mapped conversations

The following tables show the mapping between APPC verbs and CICS commands for mapped conversations. See "Return codes for APPC mapped conversations" on page 160 for details of the corresponding return code mapping.

MC_ALLOCATE	EXEC CICS ALLOCATE + EXEC CICS CONNECT PROCESS
LU_NAME(vble)	SYSID on ALLOCATE
MODE_NAME(vble)	MODENAME on ALLOCATE
TPN(vble)	PROCNAME on CONNECT PROCESS (with PROCLENGTH)
RETURN_CONTROL(WHEN_SESSION_ALLOCATED)	Default on ALLOCATE
RETURN_CONTROL(WHEN_CONWINNER_ALLOCATED)	Not supported
RETURN_CONTROL (WHEN_CONVERSATION_GROUP_ALLOCATED)	Not supported
RETURN_CONTROL(IMMEDIATE)	NOQUEUE/NOSUSPEND on ALLOCATE
SYNC_LEVEL	SYNCLEVEL on CONNECT PROCESS 0 — None 1 — Confirm 2 — Syncpoint
CONVERSATION_GROUP_ID	Not supported
SECURITY(NONE)	Not supported
SECURITY(SAME)	Default on ALLOCATE
SECURITY(PGM(USED_ID(vble)	Not supported
(PASSWORD(vble)))	Not supported
PIP(NO)	Supported by PIPLENGTH(0)
PIP(YES(vble1,vble2 ... vblen))	Supported by PIPLIST+PIPLENGTH
RESOURCE	Returned in CONVID field
RETURN_CODE	Supported

BACKOUT	EXEC CICS SYNCPOINT ROLLBACK
RETURN_CODE	Supported

MC_CONFIRM	EXEC CICS CONFIRM
RESOURCE	CONVID
RETURN_CODE	Supported
REQUEST_TO_SEND_RECEIVED	Returned in EIBSIG

MC_CONFIRMED	EXEC CICS ISSUE CONFIRMATION
RESOURCE	CONVID
RETURN_CODE	Supported

MC_DEALLOCATE	**EXEC CICS SEND LAST** + **EXEC CICS SYNCPOINT** + **EXEC CICS FREE**
RESOURCE	CONVID
TYPE(SYNC_LEVEL) None	EXEC CICS SEND LAST WAIT + EXEC CICS FREE
TYPE(SYNC_LEVEL) Confirm	EXEC CICS SEND LAST CONFIRM + EXEC CICS FREE
TYPE(SYNC_LEVEL) Syncpt	EXEC CICS SEND LAST + EXEC CICS SYNCPOINT + EXEC CICS FREE
TYPE(FLUSH)	EXEC CICS SEND LAST WAIT + EXEC CICS FREE
TYPE(CONFIRM)	EXEC CICS SEND LAST CONFIRM + EXEC CICS GDS FREE
TYPE(ABEND_PROG) Depends on setting of EIBFREE by previous command:	
EIBFREE = X'00	EXEC CICS ISSUE ABEND + EXEC CICS FREE
EIBFREE = X'FF	EXEC CICS FREE
TYPE(LOCAL)	EXEC CICS FREE
RETURN_CODE	Supported

MC_FLUSH	**EXEC CICS WAIT** or **EXEC CICS SEND WAIT**
RESOURCE	CONVID
RETURN_CODE	Supported

MC_GET_ATTRIBUTES	**EXEC CICS EXTRACT PROCESS** or **EXEC CICS ASSIGN**
RESOURCE	CONVID on EXTRACT PROCESS
SYNC_LEVEL	SYNCLEVEL on EXTRACT PROCESS 0 – None 1 – Confirm 2 – Syncpoint
PARTNER_LU_NAME	ASSIGN PRINSYSID
PARTNER_FULLY_QUALIFIED_LU_NAME	See note
MODE_NAME	See note
CONVERSATION_STATE(vble)	STATE on EXTRACT PROCESS
CONVERSATION_CORRELATOR	See note
CONVERSATION_GROUP_ID	Not supported
	Note: These values are not normally required in CICS applications and are not available at the API.
RETURN_CODE	Supported

GET_TYPE	(Examine EIBRSRCE)
RESOURCE	EIBRSRCE
TYPE(vble)	EIBRSRCE set - mapped EIBRSRCE not set - not mapped

MC_POST_ON_RECEIPT	Not supported

MC_PREPARE_FOR_SYNCPT	EXEC CICS ISSUE PREPARE
RESOURCE	CONVID
RETURN_CODE	Supported

MC_PREPARE_TO_RECEIVE	EXEC CICS SEND INVITE
TYPE(SYNC_LEVEL) none	EXEC CICS SEND INVITE WAIT
TYPE(SYNC_LEVEL) confirm	EXEC CICS SEND INVITE CONFIRM
TYPE(SYNC_LEVEL) syncpt	EXEC CICS SEND INVITE + EXEC CICS SYNCPOINT
TYPE(FLUSH)	EXEC CICS SEND INVITE WAIT
TYPE(CONFIRM)	EXEC CICS SEND INVITE CONFIRM
LOCKS(SHORT)	Defaulted
LOCKS(LONG)	Not supported
RETURN_CODE	Supported

MC_RECEIVE_AND_WAIT	EXEC CICS RECEIVE [NOTRUNCATE]
RESOURCE	CONVID field
LENGTH(vble) Input	MAXFLENGTH option
RETURN_CODE	Supported
REQUEST_TO_SEND_RECEIVED	Returned in EIBSIG
DATA	INTO or SET option
MAP_NAME	Not supported
WHAT_RECEIVED	CICS Settings
CONFIRM	EIBCONF + EIBRECV
CONFIRM_DEALLOCATE	EIBCONF + EIBFREE
CONFIRM_SEND	EIBCONF
DATA_COMPLETE	EIBCOMPL [+ EIBRECV]
DATA_INCOMPLETE	¬EIBCOMPL [+ EIBRECV}
DATA_TRUNCATED	¬EIBCOMPL if NOTRUNCATE not specified on RECEIVE
FMH_DATA_COMPLETE	EIBFMH + EIBCOMPL [+ EIBRECV]
FMH_DATA_INCOMPLETE	EIBFMH + ¬EIBCOMPL [+ EIBRECV]
FMH_DATA_TRUNCATED	EIBFMH + ¬EIBCOMPL [+ EIBRECV] if NOTRUNCATE not specified on RECEIVE
SEND	¬EIBRECV + no other flags
TAKE_SYNCPT	EIBSYNC + EIBRECV
TAKE_SYNCPT_DEALLOCATE	EIBSYNC + EIBFREE
TAKE_SYNCPT_SEND	EIBSYNC

Notes:

1. Mapping of MC_RECEIVE_AND_WAIT to EXEC CICS RECEIVE is not always one to one.

   When a CICS RECEIVE command is issued, CICS returns all the information and data (the DATA, the WHAT_RECEIVED flags, and the RETURN_CODE) at once. On completion of a CICS command, more than one indicator may be set, as shown in the WHAT_RECEIVED mapping above. It may be necessary to perform more than one subsequent command to honor the actions required by the indicators. For this reason, the action flags must be saved when they are received (because the EIB can be overwritten by subsequent CICS commands), and then acted on one by one.

   APPC does not work this way; an MC_RECEIVE_AND_WAIT verb returns either data or information about the conversation state (as indicated by WHAT_RECEIVED), but never both.

   It is necessary to program round this difference in philosophy when translating APPC verbs into CICS commands.
2. CICS EIBCOMPL settings are applicable only if NOTRUNCATE is specified on the CICS RECEIVE command.

   If NOTRUNCATE is specified, DATA_INCOMPLETE is indicated by a zero value in EIBCOMPL. CICS will save the remaining data for retrieval by subsequent RECEIVE NOTRUNCATE commands. EIBCOMPL is set when the last part of the data is passed back.

   If the NOTRUNCATE option is not specified, DATA_INCOMPLETE is indicated by the CICS LENGERR condition, and the data remaining after the RECEIVE is discarded.

MC_REQUEST_TO_SEND	EXEC CICS ISSUE SIGNAL
RESOURCE	CONVID field
RETURN_CODE	Supported

MC_SEND_DATA	EXEC CICS SEND
RESOURCE	CONVID field
DATA	FROM option
LENGTH	LENGTH option
FMH_DATA(NO)	Default
FMH_DATA(YES)	See note
MAP_NAME(NO)	Not supported
MAP_NAME(YES)	Not supported
ENCRYPT(NO)	Not supported
ENCRYPT(YES)	Not supported
RETURN_CODE	Supported
REQUEST_TO_SEND_RECEIVED	Returned in EIBSIG

Note: FMH_DATA(YES) permits the sending of LU6.1 FMHs within an APPC conversation (for example, when running a CICS program which was originally written for use on LU6.1). An LU6.1 FMH may be built either by using the EXEC CICS BUILD ATTACH command, prior to issuing the EXEC CICS SEND command, or by building the FMH within the program, putting it in the output area, and specifying the FMH option on the SEND command. Either of these two actions is equivalent to specifying FMH_DATA(YES)

MC_SEND_ERROR	EXEC CICS ISSUE ERROR
RESOURCE	CONVID field
RETURN_CODE	Supported
REQUEST_TO_SEND_RECEIVED	Returned in EIBSIG

SYNCPT	EXEC CICS SYNCPOINT
RETURN_CODE	Zero - Control returned to program. Non-zero - CICS takes action to backout the UOW (and abend the task or set EIBRLDBK).

Note: For certain specialized applications, the PREPARE flow (the first flow in syncpoint exchanges) may be sent for a particular conversation by using the command:

```
EXEC CICS ISSUE PREPARE
```

This enables any outstanding messages in the network (for example, SEND ERROR) to be received before proceeding, or deciding not to proceed, with the full syncpoint.

MC_TEST	Check EIB flags
RESOURCE	EIBRSRCE
TEST(POSTED)	Check EIB flags
TEST(REQUEST_TO_SEND_RECEIVED)	Check EIBSIG
RETURN_CODE	Not supported

WAIT	Not supported

Return codes for APPC mapped conversations

APPC RETURN_CODE	CICS return codes
OK	EIBERR zero + INVREQ not raised
ALLOCATION_ERROR	
Local allocation failures:	CICS is unable to allocate a session for an ALLOCATE command.
ALLOCATION_FAILURE_NO_RETRY	SYSIDERR raised The second and subsequent bytes of EIBRCODE give further information
ALLOCATION_FAILURE_RETRY	SYSBUSY raised if there is a HANDLE for it. Otherwise, CICS queues the request until a session is available See also the UNSUCCESSFUL return code, which relates to the NOQUEUE option on the CICS ALLOCATE command.
Remote allocation failures:	These will be returned to the program after the CONNECT PROCESS command has been issued, and the partner system has been unable to start the requested task. They may be returned on any subsequent command that relates to the session in use
CONVERSATION_TYPE_MISMATCH	TERMERR (EIBERRCD = 10086034)
PIP_NOT_ALLOWED	TERMERR (EIBERRCD = 10086031)
PIP_NOT_SPECIFIED_CORRECTLY	TERMERR (EIBERRCD = 10086032)
SECURITY_NOT_VALID	TERMERR (EIBERRCD = 080F6051)
SYNC_LEVEL_NOT_SUPPORTED_BY_PGM	TERMERR (EIBERRCD = 10086041)
SYNC_LEVEL_NOT_SUPPORTED_BY_LU	INVREQ (EIBRCODE = E000000C) Note: CICS remembers SYNC_LEVEL negotiated at bind time and does not permit a request to be sent for a sync level not supported by the remote LU.
TPN_NOT_RECOGNIZED	TERMERR (EIBERRCD = 10086021)
TRANS_PGM_NOT_AVAIL_NO_RETRY	TERMERR (EIBERRCD = 084C0000)
TRANS_PGM_NOT_AVAIL_RETRY	TERMERR (EIBERRCD = 084B6031)
BACKED_OUT	EIBSYNRB (EIBERRCD = 08240000)
DEALLOCATE_ABEND	The transaction is abended with code AZCH (EIBERRCD = 08640000)
DEALLOCATE_NORMAL	EIBFREE + ¬EIBERR
FMH_DATA_NOT_SUPPORTED	TERMERR (EIBERRCD = 08890100)
MAP_EXECUTION_FAILURE MAP_NOT_FOUND MAPPING_NOT_SUPPORTED	Not applicable. Map requests are not sent because the option is not supported.
PARAMETER_ERROR	This return code relates ONLY to the CICS ALLOCATE command. It is given when an invalid LU name or MODE name has been specified.

APPC RETURN_CODE	CICS return codes
PARAMETER_ERROR (Invalid LU name) PARAMETER_ERROR (Invalid mode name)	SYSIDERR (EIBRCODE = D0 04 .. or D0 0C ..) CBIDERR raised for invalid PROFILE on ALLOCATE command.
PROG_ERROR_NO_TRUNC PROG_ERROR_PURGING	EIBERRCD = 08890000 (RECEIVE Only) CDBERRCD = 08890000
RESOURCE_FAILURE_RETRY RESOURCE_FAILURE_NO_RETRY	EIBERRCD = A000 EIBERRCD = A000
UNSUCCESSFUL This return code relates ONLY to the APPC ALLOCATE verb with RETURN_CONTROL(IMMEDIATE) specified. This is implemented in CICS with the NOQUEUE option on the ALLOCATE command.	RETCODE = 01 04 04 Control returned to the program because a session was not immediately available.
Note: In all cases, where a value for EIBERRCD is given, EIBERR will be set to X'FF'. It is intended that the program should first test EIBERR and then examine EIBERRCD if additional information is required.	

CICS deviations from the APPC architecture

CICS allows EXEC CICS commands to be issued on APPC conversations when a backout (rollback) is required but the conversation is not in **rollback state** (state 13).

When a session is being allocated, the back-end CICS system checks the incoming bind request for valid combinations of CNOS (change number of sessions) and parallel-sessions indicators. If CICS finds that parallel-sessions is specified but CNOS is not, it sends a negative response to the bind request.

CICS allows a sync level-2 conversation to be terminated using the SEND LAST WAIT or SEND LAST CONFIRM commands. However, doing this is a deviation from the APPC architecture and should be avoided. Figure 47 on page 162 illustrates the problems that can be caused by not syncpointing a sync level-2 conversation.

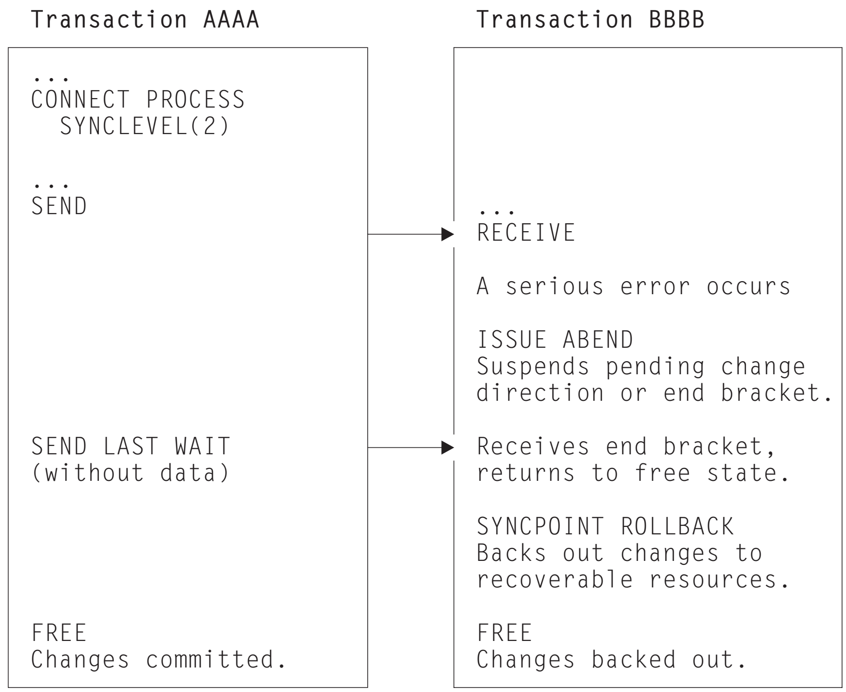

Figure 47. Losing data integrity on a sync level-2 conversation.

In this example, transaction AAAA in communicating with transaction BBBB:

1. *Transaction AAAA issues a CONNECT PROCESS command, specifying SYNCLEVEL(2).*
2. *Transaction AAAA issues a SEND command; the attach header and data is transmitted, and transaction BBBB is started.*
3. *Transaction BBBB issues a RECEIVE command.*
4. *A serious error occurs in transaction BBBB.*
5. *Transaction BBBB issues an ISSUE ABEND command. The transaction is suspended pending the receipt of a change direction or end bracket from transaction AAAA.*
6. *Transaction AAAA issues a SEND LAST WAIT command, with no data. The end bracket is transmitted to transaction BBBB.*
7. *Transaction BBBB is resumed; the incoming end bracket puts the conversation into* **free state***.*

From this point, the two transactions execute independently.

8. *Transaction AAAA frees its conversation.*
9. *Transaction AAAA ends; its changes are committed.*
10. *Transaction BBBB issues a SYNCPOINT ROLLBACK command. Its changes are backed out.*

Because transaction AAAA ends the conversation using the SEND LAST WAIT command, transaction BBBB cannot inform it that an error has occurred. The ISSUE ABEND command causes the backout-required condition to be raised in transaction BBBB; so a SYNCPOINT ROLLBACK is needed. Transaction AAAA commits changes to its resources and data integrity is lost.

The resulting state errors may also lead to the session being unbound.

Effects of CICS deviations on the transaction programmer

Where CICS deviates from the APPC architecture, there may be some effect on transaction programs running on products other than CICS and having conversations with CICS transactions. The effects can be avoided by using the following programming conventions (the verbs and return codes referred to here are described in *SNA Transaction Programmer's Reference Manual for LU Type 6.2*):

- When writing a transaction program that will converse with a CICS transaction program, do not use the verb PREPARE_TO_RECEIVE with the TYPE(CONFIRM) and LOCKS(LONG) parameters, or with the TYPE(SYNC_LEVEL) and LOCKS(LONG) when the SYNC_LEVEL is CONFIRM. Instead, use the LOCKS(SHORT) parameter to achieve the same function. The LOCKS(LONG) parameter provides only a line-flow optimization.
- When writing a transaction program that will converse with a CICS transaction program, do not depend on the distinction between the return codes PROG_ERROR_PURGING and PROG_ERROR_NO_TRUNC, and between the return codes SVC_ERROR_PURGING and SVC_ERROR_NO_TRUNC. Instead, the CICS transaction program must be coded to send additional error information after it issues the CICS EXEC ISSUE ERROR in order to describe the reason for sending the error indication.
- When writing a transaction program that will run on CICS, do not depend on the receipt of the sense data X'08890000' or X'08890100' to indicate the state of the other end of the conversation when the partner transaction program sent the error indication. Instead, the partner transaction program must be coded to send additional error information after it sends the error indication in order to describe the reason for sending the error indication.
- Because CICS may omit the negative response before an FMH-7 (ALLOCATION_ERROR), a transaction program in conversation with CICS can receive an ALLOCATION_ERROR **after** the point where the partner transaction appears to have been successfully allocated. The transaction program must therefore be written to handle this possibility.

Appendix B. Migration of LUTYPE6.1 applications to APPC links

If your installation is changing its CICS-to-CICS Intersystem communication (ISC) links from LUTYPE6.1 to APPC (LUTYPE6.2), you may want to redesign some of your existing ISC applications to take advantage of APPC function. Alternatively, you can continue to run your existing applications in “migration” mode.

The appendix contains the following topics:
- Migration mode
- “State transitions in LUTYPE6.1 migration-mode conversations” on page 167.

Migration mode

In migration mode, the front-end and back-end transactions use LUTYPE6.1 commands just as if the session was an LUTYPE6.1 session. CICS takes data from the transaction in the normal way, and formats it as an APPC mapped data stream for transmission over the link. At the receiving side, CICS analyses the APPC mapped data stream and presents the LUTYPE6.1 data and function management headers to the receiving transaction.

In general, you will not have to modify existing CICS-to-CICS ISC applications to enable them to run in migration mode on APPC links. A notable exception is the use of the ALLOCATE SESSION command. If your installation previously had individually defined ISC sessions, and your application used the ALLOCATE SESSION command to acquire a specific session, you must change this command to ALLOCATE SYSID.

The ISSUE SIGNAL command is valid for both LU types, but the WAIT SIGNAL command is available only for LUTYPE6.1.

Table 47 on page 166 compares the commands that you can use for:
- LUTYPE6.1 applications on LUTYPE6.1 links
- LUTYPE6.1 applications on APPC links (migration mode)
- APPC applications on APPC links.

As Table 47 on page 166 shows, migration mode allows you to start adding new function to an application (for example, using ISSUE ERROR or ISSUE ABEND) without converting it entirely to APPC. You can also implement different sync levels by modifying the application to use the CONNECT PROCESS command. Applications not modified to use CONNECT PROCESS will use sync level 2. The migration of an application towards the “pure” APPC level can thus be made stepwise.

To aid migration, the SESSION and CONVID options can be used interchangeably.

If a migration-mode transaction abends, the architected APPC flows take place. How this affects the connected transaction depends where the abend occurs and is often different from what you would expect if the connection were native LUTYPE6.1.

Because APPC uses different modules from LUTYPE6.1, the user exits XZCIN and XZCOUT are not taken for APPC sessions. Any programs making use of these exits on LUTYPE6.1 will need consideration.

Table 47. Migration of LUTYPE6.1 programs to APPC links

Operation	Command	LU6.1	Migration	APPC
Obtain use of a session	ALLOCATE SESSION	yes	no	no
Obtain use of a session	ALLOCATE SYSID	yes	yes	yes
Build an LUTYPE6.1 attach FMH	BUILD ATTACHID	yes	yes	no
Start a partner transaction	SEND	yes(1)	yes(4)	no
Start a partner transaction	SEND ATTACHID	yes(2)	yes(5)	no
Start a partner transaction	SEND FMH	yes(3)	yes(6)	no
Start a partner transaction	CONNECT PROCESS	no	yes(7)	yes(7)
Retrieve information about how the transaction was initiated	EXTRACT ATTACH	yes	yes	no
	EXTRACT PROCESS	no	yes	yes
Send data	SEND	yes	yes	yes
Send further LUTYPE6.1 FMHs	SEND ATTACHID	yes	yes	no
Send further LUTYPE6.1 FMHs	SEND FMH	yes	yes	no
Receive LUTYPE6.1 FMHs	EXTRACT ATTACH	yes	yes	no
Receive data	RECEIVE	yes	yes	yes
Send and receive data	CONVERSE	yes	yes	yes
Program error	ISSUE ERROR	no	yes	yes
Abend conversation	ISSUE ABEND	no	yes	yes
Request change of direction	ISSUE SIGNAL	yes	yes	yes
Await SIGNAL condition	WAIT SIGNAL	yes	no	no
Synchronize	**Level 0**	no	yes(8)	yes
Synchronize	**Level 1** SEND CONFIRM ISSUE CONFIRMATION	no no	yes(8) yes	yes yes
Synchronize	**Level 2** SEND CONFIRM ISSUE CONFIRMATION SYNCPOINT SYNCPOINT ROLLBACK	no no yes no	yes(8) yes yes yes	yes yes yes yes

Table 47. Migration of LUTYPE6.1 programs to APPC links (continued)

Operation	Command	LU6.1	Migration	APPC

Notes on migration of LUTYPE6.1 programs:

1. The CICS transaction identifier is included in the first four bytes of the data. No attach FMH generated.
2. An LUTYPE6.1 attach FMH is generated.
3. An LUTYPE6.1 FMH provided by the application program is sent.
4. An APPC attach FMH is generated, but with no TPN (TPNL=0). The CICS transaction identifier is included in the first four bytes of the data.
5. An APPC attach FMH and an LUTYPE6.1 attach FMH are generated.
6. An APPC attach FMH and an LUTYPE6.1 FMH (provided by the application program) are sent.
7. An APPC attach FMH is generated.
8. Sync levels 0 and 1 can be used if CONNECT PROCESS has been used to define the sync level in operation. If CONNECT PROCESS has not been used, sync level 2 is assumed.

State transitions in LUTYPE6.1 migration-mode conversations

In this section, the state table shows the state transitions that occur when transactions engage in LUTYPE6.1 conversations in migration mode. The state table includes the commands available and the states returned when starting a back-end transaction using the SEND [FMH|ATTACHID] command with the transaction identifier imbedded in first four bytes of user data. For back-end transactions started by CONNECT PROCESS, use the tables in “Chapter 4. State transitions in APPC mapped conversations” on page 41, but remember that the BUILD ATTACH, SEND ATTACHID, SEND FMH, and EXTRACT ATTACH commands are also available.

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 the table. The possible conversation states are shown across the top of the table. The states correspond to the columns of the table. The intersection of a row (command and EIB flag) and a column (state) represents the state transition, if any, that occurs when a particular command returning a particular EIB flag is issued in a particular 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.
×	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.

Table 48. LUTYPE6.1 conversations in migration mode, part 1

		ALLO-CATED[56]	**SEND**	**PEND-RECEIVE**	**PEND-FREE**	**RECEIVE**	**CONF-RECEIVE**
Command issued	**EIB flag returned[49]**	**State 1**	**State 2**	**State 3**	**State 4**	**State 5**	**State 6**
BUILD ATTACH	×	=	=	=	=	=	=
EXTRACT ATTACH	×	Ab[62]	Ab[62]	Ab[62]	Ab[62]	=	Ab[62]
EXTRACT PROCESS[50]	×	Ab	=	=	=	=	=
EXTRACT ATTRIBUTES	×	=	=	=	=	=	=
SEND (any valid form)	EIBERR + EIBSYNRB	Ab	13	13	13	Ab	Ab
SEND (any valid form)	EIBERR + EIBFREE	12	12	12	12	Ab	Ab
SEND (any valid form)	EIBERR	Ab	5	5	5	Ab	Ab
SEND INVITE WAIT	×	5	5	Ab	Ab	Ab	Ab
SEND INVITE CONFIRM	×	5	5	Ab	Ab	Ab	Ab
SEND INVITE	×	3	3	Ab	Ab	Ab	Ab
SEND LAST WAIT	×	12	12	Ab	Ab	Ab	Ab
SEND LAST CONFIRM	×	12	12	Ab	Ab	Ab	Ab
SEND LAST	×	4	4	Ab	Ab	Ab	Ab
SEND WAIT	×	2	=	Ab	Ab	Ab	Ab
SEND CONFIRM	×	2	=	5[59]	12[59]	Ab	Ab
SEND	×	2	=	Ab	Ab	Ab	Ab
RECEIVE	EIBERR + EIBSYNRB	Ab	13[52]	13[55]	Ab	13	Ab
RECEIVE	EIBERR + EIBFREE	Ab	12[52]	12[55]	Ab	12	Ab
RECEIVE	EIBERR	Ab	5[52]	5[55]	Ab	=	Ab
RECEIVE	EIBSYNC + EIBFREE	Ab	11[52]	11[55]	Ab	11	Ab
RECEIVE	EIBSYNC + EIBRECV	Ab	9[52]	9[55]	Ab	9	Ab
RECEIVE	EIBSYNC	Ab	10[52]	10[55]	Ab	10	Ab
RECEIVE	EIBCONF + EIBFREE	Ab	8[52]	8[55]	Ab	8	Ab
RECEIVE	EIBCONF + EIBRECV	Ab	6[52]	6[55]	Ab	6	Ab
RECEIVE	EIBCONF	Ab	7[52]	7[55]	Ab	7	Ab
RECEIVE	EIBFREE	Ab	12[52]	12[55]	Ab	12	Ab
RECEIVE	EIBRECV	Ab	5[52]	5[55]	Ab	=	Ab
RECEIVE NOTRUNCATE[51]	EIBCOMPL[51]	Ab	5[52]	5[55]	Ab	=	Ab
RECEIVE	×	Ab	=[52]	2[55]	Ab	2	Ab

Note: See page 170 for footnotes.

Table 49. LUTYPE6.1 conversations in migration mode, part 2

CONF-SEND	CONF-FREE	SYNC-RECEIVE	SYNC-SEND	SYNC-FREE	FREE	ROLL-BACK	
State 7	State 8	State 9	State 10	State 11	State 12	State 13	Command returns
=	=	=	=	=	=	=	Immediately
Ab[62]	Ab[62]	Ab[62]	Ab[62]	Ab[62]	Ab[62]	Ab[62]	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
Ab	Ab	Ab	Ab	Ab	Ab	Ab	After confirm flow detected
Ab	Ab	Ab	Ab	Ab	Ab	Ab	After error flow detected
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

table continued ......

Table 50. LUTYPE6.1 conversations in migration mode, part 3

Command issued	EIB flag returned[49]	ALLO-CATED[56]	SEND	PEND-RECEIVE	PEND-FREE	RECEIVE	CONF-RECEIVE
		State 1	State 2	State 3	State 4	State 5	State 6
CONVERSE[53]	EIB flags and states as for RECEIVE						
ISSUE CONFIRMATION	×	Ab	Ab	Ab	Ab	Ab	5
ISSUE ERROR	EIBFREE	Ab	12	12	Ab	12	12
ISSUE ERROR	×	Ab	=	2	Ab	2	2
ISSUE ABEND	×	Ab	12	12	12	12	12
ISSUE SIGNAL[57]	×	Ab	=	=	Ab	=	=
ISSUE PREPARE	EIBERR + EIBSYNRB	Ab[62]	13	13	13	Ab[62]	Ab[62]
ISSUE PREPARE	EIBERR + EIBFREE	Ab[62]	12	12	12	Ab[62]	Ab[62]
ISSUE PREPARE	EIBERR	Ab[62]	5	5	5	Ab[62]	Ab[62]
ISSUE PREPARE	×	Ab[62]	10[64]	9[64]	11[64]	Ab[62]	Ab[62]
SYNCPOINT[60]	EIBRLDBK	=	2 or 5[61]	2 or 5[61]	2 or 5[61]	Ab[63]	Ab[63]
SYNCPOINT[60]	×	=	=	5	12	Ab[63]	Ab[63]
SYNCPOINT ROLLBACK[60]	×	=	2 or 5[61]	2 or 5[61]	2 or 5[61]	2 or 5[61]	2 or 5[61]
WAIT	×	Ab	=	5	12	Ab	Ab
FREE	×	End	End[54]	Ab	End	Ab	Ab

49. 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 occur 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.
50. You can issue the EXTRACT PROCESS command from the back-end transaction only.
51. 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 small to contain all the data received, CICS truncates the data and sets the LENGERR condition.
52. Equivalent to SEND INVITE WAIT followed by RECEIVE.
53. Equivalent to SEND INVITE WAIT [FROM] followed by RECEIVE.
54. Equivalent to SEND LAST WAIT followed by FREE.
55. Equivalent to WAIT followed by RECEIVE.
56. 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**.
57. ISSUE SIGNAL sets the partner's EIBSIG flag.
58. The back-end transaction starts in **receive state**.
59. No data may be included with SEND CONFIRM.

Table 51. LUTYPE6.1 conversations in migration mode, part 4

CONF-SEND	**CONF-FREE**	**SYNC-RECEIVE**	**SYNC-SEND**	**SYNC-FREE**	**FREE**	**ROLL-BACK**	
State 7	**State 8**	**State 9**	**State 10**	**State 11**	**State 12**	**State 13**	**Command returns**
States as for RECEIVE							When data available
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
=	=	=	=	=	Ab	Ab	Immediately
Ab[62]	Ab[62]	Ab[62]	Ab[62]	Ab[62]	Ab[62]	Ab[62]	After response from partner
Ab[62]	Ab[62]	Ab[62]	Ab[62]	Ab[62]	Ab[62]	Ab[62]	After error detected
Ab[62]	Ab[62]	Ab[62]	Ab[62]	Ab[62]	Ab[62]	Ab[62]	After error detected
Ab[62]	Ab[62]	Ab[62]	Ab[62]	Ab[62]	Ab[62]	Ab[62]	After response from partner
Ab	Ab	2 or 5[61]	2 or 5[61]	2 or 5[61]	=	Ab	After response from partner
Ab	Ab	2	2	12	=	Ab	After response from partner
2 or 5[61]	2 or 5[61]	2 or 5[61]	2 or 5[61]	2 or 5[61]	=	2 or 5[61]	After rollback across UOW
Ab	Ab	Ab	Ab	Ab	Ab	Ab	Immediately
Ab	Ab	Ab	Ab	Ab	End	Ab	Immediately

60. 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, including MRO conversations. For the SYNCPOINT command, all these conversations must be in **send state**.

61. The state of each conversation after rollback depends on several factors:
    - The system you are communicating with. Some earlier versions of CICS handle rollback differently from CICS Transaction Server for z/OS, Version 2 Release 2.
    - The conversation state at the last syncpoint, or at the beginning of the conversation if there was no previous sync point. This state is the one adopted according to the APPC architecture. CICS Transaction Server for z/OS, Version 2 Release 2 follows the architecture.

62. This results, not in an ATCV abend, but in an INVREQ return code.

63. This causes an ASP2 abend, not an ATCV.

64. 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.

Appendix C. Differences between APPC mapped and MRO conversations

When a SEND command is issued on an MRO session, CICS does not defer sending the data, so control indicators cannot be added to the data after a SEND command has been issued. The same command sequence may therefore require more flows on an MRO session than it does on an APPC session but, if the receiving transaction is correctly designed to be driven by the conversation state, the same effects are achieved.

Different treatment of command sequences

Some of the differences between APPC mapped and MRO conversations are shown in the command sequence in Table 52.

Table 52. How the same command sequence operates differently in APPC mapped and MRO conversations

Commands	APPC mapped	MRO
`EXEC CICS SEND CONVID(REM1) FROM(data1) LENGTH(251)`	sending is deferred	`data1` is sent
`EXEC CICS SYNCPOINT`	syncpoint request added to `data1`, and both are sent	syncpoint request is sent with null data
`EXEC CICS SEND CONVID(REM1) FROM(data2) LENGTH(251) INVITE`	sending of `data2`, with `INVITE`, is deferred	`data2` with `INVITE` is sent
`EXEC CICS WAIT CONVID(REM1)`	`data2`, with `INVITE`, is sent	(nothing to send)
`EXEC CICS RECEIVE CONVID(REM1) . . (INVITE received)`		
`EXEC CICS SEND CONVID(REM1) FROM(data3) LENGTH(251) LAST`	sending of `data3`, with `LAST` indicator, is deferred	`data3` is sent, but without `LAST` indicator
`EXEC CICS SYNCPOINT`	syncpoint request and `LAST` indicator added to `data3` and sent	syncpoint request and `LAST` indicator are sent with null data

The WAIT option can, of course, be added to the SEND command to cause immediate transmission on APPC links; for example:

```
SEND CONVID(REM1)
     FROM(data2)
     LENGTH(251)
     INVITE
     WAIT
RECEIVE SESSION(REM1)
```

There are no significant differences between the MRO and APPC mapped implementations of this command sequence. However, with MRO, a SEND command with the WAIT option causes CICS to suspend the transaction until the partner system has received the data.

Unlike APPC, MRO allows only one outstanding SEND to be transmitted. This means that when a transaction issue two successive SEND commands (without the WAIT option) to transmit data, the second piece of data does not flow until the partner system has received the first.

A further implementation difference arises between APPC mapped and MRO for command sequences that contain an implicit change of direction. For MRO, a RECEIVE command must not be issued unless the conversation is in **receive state** (state 5).

Using the LAST option

The LAST option on the SEND command indicates the end of the conversation. No further data flows can occur on the session, and the next action must be to free the session. However, the session can still carry CICS syncpointing flows before it is freed, provided the LAST request has not been flushed.

A syncpoint, whether on an APPC or MRO session, is initiated explicitly by a SYNCPOINT command, or implicitly by a RETURN command. However, the circumstances under which session syncpointing occurs, and the ways in which syncpointing can be avoided on the session, differ for APPC and MRO.

The LAST option and syncpoint flows on APPC sessions

If an APPC mapped conversation has been terminated by a SEND LAST command, without the WAIT option, transmission will have been deferred, and the syncpointing activity causes the final transmission to occur with an added syncpoint request. The conversation is thus automatically involved in the syncpoint.

If the conversation is not to be involved in the syncpoint (for example, because the partner transaction does not access any recoverable resources), the transaction must issue a SEND LAST WAIT command, or a FREE command, to force the transmission before using a command that causes a syncpoint.

The LAST option and syncpoint flows on MRO sessions

If an MRO conversation is terminated by a SEND LAST command, without the WAIT option, the WAIT implicit in all MRO commands is applied, and the data is transmitted. However, in anticipation of subsequent syncpoint flows, CICS does not send the LAST indicator with this data.

If the conversation is not to be involved in the syncpoint (for example, because the partner transaction does not access any recoverable resources) you must specify the WAIT option explicitly on the SEND LAST command to force the LAST indicator to be sent with the data. Alternatively, you could follow the SEND LAST command by a FREE command.

Appendix D. Below the SNA interface

The information provided in the main chapters of this book enables a programmer to construct valid command sequences for distributed processes. However, to design high-performance distributed processes, you need some understanding of the SNA protocols and corresponding data flow control (DFC) indicators that CICS uses for DTP. You also need to understand how the DFC indicators relate to the CICS commands and options. In addition, you need this knowledge to understand the CICS trace.

Except for some commands that can cause transmissions "against the flow" (such as ISSUE SIGNAL), the conversation flow and indicators set are dictated by the transaction currently in **send state** (state 2).

SNA indicators and records

SNA indicators and records can be generated either explicitly as a result of a CICS command, or automatically when CICS detects that they are needed. The most common SNA indicators and records are described below:

Begin_bracket and conditional_end_bracket
The begin_bracket (BB) and condition_end_bracket (CEB) indicators in the request header (RH) denote respectively the beginning and end of a conversation between two transactions. Because the BB is generated automatically at the start of a conversation, you need only consider the CEB. The CEB is generated by a SEND with the LAST option, an ISSUE ABEND, a FREE command, or task termination before the conversation is ended.

Function management headers
Function management headers (FMHs) are records sent on a conversation which contain SNA control data. Several types of FMH are defined under SNA; but only two (FMH5 and FMH7) are relevant to APPC DTP.

The FMH5, also known as the attach FMH, is sent with BB and contains the information required to initiate the back-end transaction.

The FMH7 is issued by the ISSUE ERROR, ISSUE ABEND, and SYNCPOINT ROLLBACK commands. In addition, if the back-end system rejects the FMH5, an FMH7 is sent to the front-end transaction. The FMH7 contains a 4-byte code, called the sense code, which describes the error. This code is set in EIBERRCD (or CDBERRCD for basic conversations). The FMH7 may be followed by log data. This log data is included in message DFHZN2701 on the sending system and DFHZC3433 on the receiving system.

Change direction
The change direction (CD) indicator, found in the RH, switches the issuing transaction from **send state** (state 2) to **receive state** (state 5). CD is generated explicitly by either of the following:

- A SEND command with the INVITE option
- A CONVERSE command.

PS header (type 10)
PS headers (type 10) are records sent on a conversation which contain syncpoint requests. These headers contain a 2-byte syncpoint request code (for example, *prepare, request commit, committed, and forget*). In addition, the initial record sent contains a 2-byte modifier specifying the conversation state after a successful syncpoint exchange.

Request mode and responses

When data is sent, a response confirming receipt of the data is not normally expected. This is because data is normally sent in RQE (request exception response) mode, meaning that a response is required only if an error condition needs to be transmitted. This response is called −RSP (negative response) and might precede an FMH7. However, if data is sent with the CONFIRM option, the data is sent in RQD (request definite response) mode. This means that the sending transaction will suspend until a DR (definite response) or -RSP is received. The partner transaction generates a DR with the ISSUE CONFIRMATION command.

When SNA indicators are transmitted

To optimize the use of ISC sessions, CICS defers output processing for SEND commands. Deferred output often enables CICS to add SNA indicators to waiting data before transmitting it. The number of transmissions on the session is thereby reduced.

For APPC sessions, this reduction is achieved by accumulating as much data as possible in a CICS buffer before actually transmitting it across the link. Thus the data from a series of SEND commands is transmitted only when the buffer becomes full or when transmission must be forced (for example, if SEND WAIT is encountered).

Optimization of ISC transmission does not affect the number of data flows that the application programming interface sees.

For more information on the APPC protocol, see the *SNA LU6.2 Reference: Peer Protocols* book, SC31-6808.

Glossary

This glossary contains definitions of those terms and abbreviations that relate specifically to the contents of this book. It also contains terms and definitions from the *IBM Dictionary of Computing*, published by McGraw-Hill.

If you do not find the term you are looking for, refer to the Index or to the *IBM Dictionary of Computing*.

ACB. Access method control block (VTAM).

ACF/NCP/VS. Advanced Communication Facilities/Network Control Program/Virtual Storage.

ACF/VTAM. Advanced Communication Facilities, Virtual Telecommunications Access Method. A set of programs that control communication between terminals and application programs running under VSE, OS/VS1, and MVS.

Advanced Program-to-Program Communication (APPC). The general term chosen for the LUTYPE6.2 protocol under Systems Network Architecture (SNA).

alternate facility. An IRC or SNA session that is obtained by a transaction by means of an ALLOCATE command. Contrast with principal facility.

APPC. Advanced Program-to-Program Communication.

ATI. Automatic transaction initiation.

attach header. In SNA, a function management header that causes a remote process or transaction to be attached.

back-end transaction. In synchronous transaction-to-transaction communication, a transaction that is started by a front-end transaction.

backout. See dynamic transaction backout.

bind. In SNA products, a request to activate a session between two logical units.

CDB. Conversation data block.

central processing complex (CPC). A single physical processing system, such as the whole of an ES/9000® 9021 Model 820, or one physical partition of such a machine. A physical processing system consists of main storage, and one or more central processing units (CPUs), time-of-day (TOD) clocks, and channels, which are in a single configuration. A CPC also includes channel subsystems, service processors, and expanded storage, where installed.

CICSplex. (1) A CICS complex. A CICSplex consists of two or more regions that are linked using CICS intercommunication facilities. The links can be either intersystem communication (ISC) or multiregion operation (MRO) links, but within a CICSplex are more usually MRO. Typically, a CICSplex has at least one terminal-owning region (TOR), more than one application-owning region (AOR), and may have one or more regions that own the resources that are accessed by the AORs. (2) The largest set of CICS regions or systems to be manipulated by a single CICSplex SM entity.

CICSPlex System Manager (CICSPlex SM). An IBM CICS system-management product that provides a single-system image and a single point of control for one or more CICSplexes.

class of service (COS). An ACF/VTAM® facility that allows APPC sessions to have different characteristics to provide a user with alternate routing, mixed traffic, and trunking. Based on their class of service, sessions can take different virtual routes, use different physical links, and be of high or low priority to suit the traffic carried on them.

Common Programming Interface (CPI). An SAA standard specifying the languages, commands, and calls that can be used in an SAA application program.

conversation. A sequence of exchanges between two transactions over a session, delimited by SNA brackets.

conversation data block. An area used by a program to obtain information about the outcome of a DTP command on an APPC basic conversation.

COS. Class of service.

CPC. Central processing complex.

CPI. Common Programming Interface.

conversation. A sequence of exchanges between transactions over a session, delimited by SNA brackets.

cross-system coupling facility (XCF). The MVS/ESA cross-system coupling facility provides the services that are needed to join multiple MVS images into a sysplex. XCF services allow authorized programs in a multisystem environment to communicate (send and receive data) with programs in the same, or another, MVS image. Multisystem applications can use the services of XCF, including MVS components and application subsystems (such as CICS), to communicate across a sysplex. See the *MVS/ESA Planning: Sysplex Management* manual, GC28-1620, for more information about the use of XCF in a sysplex.

data link protocol. A set of rules for data communication over a data link in terms of a transmission code, a transmission mode, and control and recovery procedures.

data security. Prevention of access to or use of stored information without authorization.

distributed transaction processing (DTP). The distribution of processing between transactions that communicate synchronously with one another over intersystem or interregion links.

DL/I. Data Language/I. An IBM database management facility.

DTP. Distributed transaction processing.

dynamic transaction backout. The process of canceling changes made to stored data by a transaction following the failure of that transaction for whatever reason.

EIB. EXEC interface block.

FMH. Function management header.

front-end transaction. In synchronous transaction-to-transaction communication, the transaction that acquires the session to another system and initiates a transaction on that system. Contrast with back-end transaction.

function management header (FMH). In SNA, one or more headers optionally present in the leading request unit (RU) of an RU chain. It allows one session partner in a LU-LU session to send function management information to the other.

function shipping. The process, transparent to the application program, by which CICS accesses resources when those resources are actually held on another CICS system.

GDS. Generalized data stream.

generalized data stream (GDS). The data stream used for conversations on APPC sessions.

host computer. The primary or controlling computer in a data communication system.

IMS/VS. Information Management System/Virtual Storage.

inquiry. A request for information from storage.

intercommunication facilities. A generic term covering intersystem communication (ISC) and multiregion operation (MRO).

interregion communication (IRC). The method by which CICS implements multiregion operation (MRO).

intersystem communication (ISC). Communication between separate systems by means of SNA networking facilities or by means of the application-to-application facilities of VTAM.

IRC. Interregion communication.

ISC. Intersystem communication.

local resource. In CICS intercommunication, a resource that is owned by the local system.

local system. In CICS intercommunication, the CICS system from whose point of view intercommunication is being discussed.

logical unit (LU). A port through which a user gains access to the services of a network.

LU. Logical unit.

LU-LU session. A session between two logical units in an SNA network.

modegroup. A VTAM LOGMODE entry which can specify (among other things) the class of service required for a group of APPC sessions.

modename. The name of a modeset.

modeset. A group of APPC sessions specified in CICS.

MRO. Multiregion operation.

multiregion operation (MRO). Communication between CICS systems without the use of SNA networking facilities. The systems must be in the same operating system; or, if the XCF access method is used, in the same MVS sysplex.

multitasking. Concurrent execution of application programs within a CICS partition or region.

multithreading. Use, by several transactions, of a single copy of an application program.

MVS. Multiple Virtual Storage. An alternative name for OS/VS2 Release 3, or MVS/ESA.

MVS image. A single occurrence of the MVS/ESA operating system that has the ability to process a workload. One MVS image can occupy the whole of a CPC, or one physical partition of a CPC, or one logical partition of a CPC that is operating in PR/SM™ mode.

MVS sysplex. See *sysplex*.

network. A configuration connecting two or more terminal installations.

network configuration. In SNA, the group of links, nodes, machine features, devices, and programs that make up a data processing system, a network, or a communication system.

Operating System/Virtual Storage (OS/VS). A compatible extension of the IBM System/360 Operating System that supports relocation hardware and the extended control facilities of System/360.

OS/VS. Operating System/Virtual Storage.

PIP. Program initialization parameters.

principal facility. The terminal or logical unit that is connected to a transaction at its initiation. Contrast with alternate facility.

program initialization parameters (PIP). Specially formatted data passed to a back-end transaction with the CONNECT PROCESS command.

queue. A line or list formed by items in a system waiting for service; for example, tasks to be performed or messages to be transmitted in a message-switching system.

RACF. The Resource Access Control Facility program product. An external security management facility.

region. A section of the dynamic area that is allocated to a job step or system task. In this manual, the term is used to cover partitions and address spaces in addition to regions.

remote resource. In CICS intercommunication, a resource that is owned by a remote system.

remote system. In CICS intercommunication, a system that the local CICS system accesses via intersystem communication or multiregion operation.

resource. Any facility of the computing system or operating system required by a job or task, and including main storage, input/output devices, the processing unit, data sets, and control or processing programs.

rollback. A programmed return to a prior checkpoint. In CICS, the cancelation by an application program of the changes it has made to all recoverable resources during the current unit of work.

RU. Request/response unit. In SNA, the basic unit of information entering or leaving the transmission subsystem. It may contain data, acknowledgements, control commands, or responses to commands.

SAA. Systems Application Architecture.

security. Prevention of access to or use of data or programs without authorization.

session. In CICS intersystem communication, an SNA LU-LU session.

SNA. Systems Network Architecture.

subsystem. A secondary or subordinate system.

synchronization level. The level of synchronization (0, 1, or 2) established for an APPC session.

syncpoint. Synchronization point. During transaction processing, a reference point to which protected resources can be restored if a failure occurs.

sync level. synchronization level.

sysplex. A systems complex, consisting of multiple MVS images coupled together by hardware elements and software services. When multiple MVS images are coupled using XCF, which provides the services to form a sysplex, they can be viewed as a single entity.

system. In CICS, an assembly of hardware and software capable of providing the facilities of CICS for a particular installation.

Systems Application Architecture (SAA). A set of common standards and procedures for working with IBM systems and data.

Systems Network Architecture (SNA). The description of the logical structure, formats, protocols, and operational sequences for transmitting information units through, and controlling the configuration and operation of, networks. The structure of SNA allows the end users to be independent of, and unaffected by, the specific facilities used for information exchange.

task. (1) A unit of work for the processor; therefore the basic multiprogramming unit under the control program. (CICS runs as a task under VSE, OS/VS, MVS, or MVS/ESA.) (2) Under CICS, the execution of a transaction for a particular user. Contrast with transaction.

TCAM. Telecommunications Access Method.

TCT. Terminal control table.

temporary storage control. The CICS element that provides temporary data storage facilities.

temporary storage table. A table describing temporary storage queues and queue prefixes for which CICS is to provide recovery.

terminal. In CICS, a device equipped with a keyboard and some kind of display, capable of sending and receiving information over a communication channel.

terminal control. The CICS element that controls all CICS terminal activity.

terminal control table (TCT). A table describing a configuration of terminals, logical units, or other CICS systems in a CICS network with which the CICS system can communicate.

terminal operator. The user of a terminal.

transaction. A transaction can be regarded as a unit of processing (consisting of one or more application programs) initiated by a single request, often from a terminal. A transaction may require the initiation of one or more tasks for its execution. Contrast with task.

transaction backout. The cancelation, as a result of a transaction failure, of all updates performed by a task.

transaction identifier. Synonym for transaction name. For example, a group of up to four characters entered by an operator when selecting a transaction.

transaction restart. The restart of a task after a transaction backout.

transaction routing. A CICS facility that allows terminals or logical units connected to one CICS region to initiate and to communicate with transactions in another CICS region within the same processor system or in another CICS system connected by an APPC link.

transient data control. The CICS element that controls sequential data files and intrapartition data.

unit of work (UOW). A sequence of actions that can be regarded as logically-related for the purposes of CICS error recovery mechanisms.

UOW. Unit of work.

VTAM. See ACF/VTAM.

XCF. Cross-system coupling facility.

Bibliography

CICS Transaction Server for z/OS

CICS Transaction Server for z/OS Release Guide	GC34-5983
CICS Transaction Server for z/OS Migration Guide	GC34-5984
CICS Transaction Server for z/OS Installation Guide	GC34-5985
CICS Transaction Server for z/OS Program Directory	GI10-2543
CICS Transaction Server for z/OS Licensed Program Specification	GC34-5987

The above titles are the only books provided automatically in hardcopy with CICS Transaction Server for z/OS, Version 2 Release 2. Several other books are available to order in hardcopy. Further information about the forms in which the published information for CICS is delivered may be found in *CICS Transaction Server for z/OS Release Guide*, or *CICS Transaction Server for z/OS Installation Guide*.

CICS books for CICS Transaction Server for z/OS

General	
CICS User's Handbook	SC34-5986
CICS Transaction Server for z/OS Glossary	GC34-5696
Administration	
CICS System Definition Guide	SC34-5988
CICS Customization Guide	SC34-5989
CICS Resource Definition Guide	SC34-5990
CICS Operations and Utilities Guide	SC34-5991
CICS Supplied Transactions	SC34-5992
Programming	
CICS Application Programming Guide	SC34-5993
CICS Application Programming Reference	SC34-5994
CICS System Programming Reference	SC34-5995
CICS Front End Programming Interface User's Guide	SC34-5996
CICS C++ OO Class Libraries	SC34-5997
CICS Distributed Transaction Programming Guide	SC34-5998
CICS Business Transaction Services	SC34-5999
Java™ Applications in CICS	SC34-6000
Diagnosis	
CICS Problem Determination Guide	SC34-6002
CICS Messages and Codes	GC34-6003
CICS Diagnosis Reference	LY33-6099
CICS Data Areas	LY33-6100
CICS Trace Entries	SC34-6004
CICS Supplementary Data Areas	LY33-6101
Communication	
CICS Intercommunication Guide	SC34-6005
CICS Family: Interproduct Communication	SC34-6030
CICS Family: Communicating from CICS on System/390	SC34-6031
CICS External Interfaces Guide	SC34-6006
CICS Internet Guide	SC34-6007
Special topics	
CICS Recovery and Restart Guide	SC34-6008
CICS Performance Guide	SC34-6009

CICS IMS Database Control Guide	SC34-6010
CICS RACF Security Guide	SC34-6011
CICS Shared Data Tables Guide	SC34-6012
CICS Transaction Affinities Utility Guide	SC34-6013
CICS DB2 Guide	SC34-6014

CICSPlex SM books for CICS Transaction Server for z/OS

General	
CICSPlex SM Concepts and Planning	SC34-6015
CICSPlex SM User Interface Guide	SC34-6016
CICSPlex SM Commands Reference Summary	SX33-6119
CICSPlex SM Web User Interface Guide	SC34-6018
Administration and Management	
CICSPlex SM Administration	SC34-6019
CICSPlex SM Operations Views Reference	SC34-6020
CICSPlex SM Monitor Views Reference	SC34-6021
CICSPlex SM Managing Workloads	SC34-6022
CICSPlex SM Managing Resource Usage	SC34-6023
CICSPlex SM Managing Business Applications	SC34-6024
Programming	
CICSPlex SM Application Programming Guide	SC34-6025
CICSPlex SM Application Programming Reference	SC34-6026
Diagnosis	
CICSPlex SM Resource Tables Reference	SC34-6027
CICSPlex SM Messages and Codes	SC34-6028
CICSPlex SM Problem Determination	GC34-6029

Other CICS books

Designing and Programming CICS Applications	SR23-9692
CICS Application Migration Aid Guide	SC33-0768
CICS Family: API Structure	SC33-1007
CICS Family: Client/Server Programming	SC33-1435
CICS Transaction Gateway for OS/390 Administration	SC34-5528
CICS Family: General Information	GC33-0155
CICS 4.1 Sample Applications Guide	SC33-1173
CICS/ESA 3.3 XRF Guide	SC33-0661

Note: The *CICS Transaction Server for OS/390: Planning for Installation* book that was part of the library for CICS Transaction Server for OS/390, Version 1 Release 3, is now merged with the *CICS Transaction Server for z/OS Installation Guide*. If you have any questions about the CICS Transaction Server for z/OS library, see *CICS Transaction Server for z/OS Installation Guide* which discusses both hardcopy and softcopy books and the ways that the books can be ordered.

Books from related libraries

IMS

- *CICS/VS to IMS/VS Intersystem Communication Primer*, SH19-6247 through SH19-6254

- *IMS/ESA Data Communication Administration Guide*, SC26-3060
- *IMS/ESA Operations Guide*, SC26-8029

Systems Application Architecture (SAA)

- *An Overview*, GC26-4341
- *Common Programming Interface: COBOL Reference*, SC26-4354
- *Common Programming Interface Communications*, SC26-4399
- *Common Programming Interface: C Reference*, SC26-4353
- *SAA Common Programming Interface Resource Recovery Reference*, SC31-6821
- *Writing Applications: a Design Guide*, SC26-4362

Systems Network Architecture (SNA)

- *Concepts and Products*, GC30-3072
- *Format and Protocol Reference Manual: Architecture Logic*, SC30-3112
- *Format and Protocol Reference Manual: Architecture Logic for LU Type 6.2*, SC30-3269
- *Format and Protocol Reference Manual: Distribution Services*, SC30-3098
- *Formats*, GA27-3136
- *LU6.2 Reference: Peer Protocols*, SC31-6808
- *LU6.2 Reference: Verb Descriptions*, GC30-3084
- *Sessions Between Logical Units*, GC20-1868
- *Technical Overview*, GC30-3073

Determining if a publication is current

IBM regularly updates its publications with new and changed information. When first published, both hardcopy and BookManager® softcopy versions of a publication are usually in step. However, due to the time required to print and distribute hardcopy books, the BookManager version is more likely to have had last-minute changes made to it before publication.

Subsequent updates will probably be available in softcopy before they are available in hardcopy. This means that at any time from the availability of a release, softcopy versions should be regarded as the most up-to-date.

For CICS Transaction Server books, these softcopy updates appear regularly on the *Transaction Processing and Data Collection Kit* CD-ROM, SK2T-0730-xx. Each reissue of the collection kit is indicated by an updated order number suffix (the -xx part). For example, collection kit SK2T-0730-06 is more up-to-date than SK2T-0730-05. The collection kit is also clearly dated on the cover.

Updates to the softcopy are clearly marked by revision codes (usually a "#" character) to the left of the changes.

Accessibility

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use software products successfully.

You can perform most tasks required to set up, run, and maintain your CICS® system in one of these ways:

- using a 3270 emulator logged on to CICS
- using a 3270 emulator logged on to TSO
- using a 3270 emulator as an MVS™ system console

IBM® Personal Communications (Version 5.0.1 for Windows® 95, Windows 98, Windows NT® and Windows 2000; version 4.3 for OS/2) provides 3270 emulation with accessibility features for people with disabilities. You can use this product to provide the accessibility features you need in your CICS system.

Index

A

abend codes
  ASP1 66, 135
  ASP2 49, 171
  ASP3 131, 133, 134
  ASPN 121
  ATCV 41, 113, 167
  AZI1 65
abnormal termination
  APPC basic conversations 80
  APPC mapped conversations 32, 34
  LUTYPE6.1 conversations 108
  MRO conversations 60
ALLOCATE command
  APPC basic conversations 86, 91
  APPC mapped conversations 23, 40, 48
  LUTYPE6.1 conversations 105, 109
  MRO conversations 53, 62
  PARTNER option 23
allocating a session
  APPC basic conversations 71
  APPC mapped conversations 23
  using ATI 23, 71
alternate facility 7
API (application programming interface) 16, 17
APPC
  data stream 75
  generalized data stream 75
  mapping to APPC architecture 147
APPC architecture
  CICS mapping 148, 155
  CICS mapping to 147
  deviations 161
APPC basic conversations
  abnormal termination 80, 81, 83
  application programming 71
  back-end transaction 71, 73, 100
  CDB data 77
  CICS mapping to APPC verbs 148
  CONFIRM option
    GDS SEND command 76
  CONVDATA fields 86
  conversation data block (CDB) 86
  ending one 83
  flushing a CICS buffer 76
  front-end transaction 71
  GDS ALLOCATE command 86, 91, 100
  GDS ASSIGN command 73
  GDS CONNECT PROCESS command 72, 91
  GDS EXTRACT PROCESS command 73, 91
  GDS FREE command 83, 84, 92
  GDS ISSUE ABEND command 80
  GDS ISSUE ERROR command 81
  GDS ISSUE SIGNAL command 80
  GDS RECEIVE command 77, 88
  GDS SEND command 75, 76
  GDS WAIT command 76
APPC basic conversations *(continued)*
  INVITE option 76
  RETCODE values 85
  session data and error codes 85
  starting 71
  state transitions 93
APPC mapped conversations
  abnormal termination 32
  ALLOCATE command 23, 40, 48
  ASSIGN command 25
  attaching partner transactions 24
  CICS mapping to APPC verbs 155
  CONNECT PROCESS command 24, 40
  CONVERSE command 31, 32, 38, 40
  ending one 35
  EXTRACT PROCESS command 25, 26, 40
  FREE command 35, 36, 40
  front-end transaction 23
  ISSUE ABEND command 32
  ISSUE CONFIRMATION command 33
  ISSUE ERROR command 32
  RECEIVE command 38
  SEND command 27, 56
  starting 23
application design 8, 11
application program development
  APPC basic conversations 71
  APPC mapped conversations 23
  LUTYPE6.1 conversations 105
  MRO conversations 53
application programming
  APPC basic conversations 71, 77
  APPC mapped conversations 23
  CICS mapping to APPC verbs 147
  LUTYPE6.1 conversations 105, 113
  MRO conversations 53
  MRO distributed transaction processing 53, 65
application programming interface (API) 16, 17
ASP1 abend 135
  MRO conversations 66
ASP2 abend
  APPC mapped conversations 49
  LUTYPE6.1 conversations 171
ASP3 abend 131, 133, 134
ASPN abend 121
assembler language 17, 69, 87
ASSIGN command
  APPC basic conversations 73
  APPC mapped conversations 25
  MRO conversations 56
asynchronous processing 3
ATCV abend
  APPC mapped conversations 41
  LUTYPE6.1 conversations 113, 167
ATI (automatic transaction initiation) 17
  APPC basic conversations 71
  APPC mapped conversations 23
  LUTYPE6.1 conversations 105, 116

ATI (automatic transaction initiation) *(continued)*
MRO conversations 53
attach request 7
attaching partner transactions
APPC basic conversations 72
APPC mapped conversations 24
LUTYPE6.1 conversations 105
MRO conversations 54
automatic transaction initiation (ATI) 17
APPC basic conversations 71
APPC mapped conversations 23
LUTYPE6.1 conversations 105, 116
MRO conversations 53
AZI1 abend
MRO conversations 65

B

back-end transaction 6, 14
APPC basic conversations 71, 73, 100, 102
EXTRACT PROCESS command 48
failure to start 106
LUTYPE6.1 conversations 105, 165
MRO conversations 53
backing out changes 136, 139
performance effect 8
to recoverable resources 8, 120
backout 120, 136, 139
effect on performance 8
of recoverable resources 8
basic conversations 71
command sequences 91
CONVDATA fields 86
conversation design 77
RETCODE values 85
session data and error codes 85
state transitions 93
structured fields 77
BUFFER option
GDS RECEIVE command 79
BUILD ATTACH command
LUTYPE6.1 conversations 106, 112
MRO conversations 54, 63

C

C language 17, 69, 87
CDB data 18, 77, 82, 85
checking the conversation state of a transaction 30
CICS mapping to APPC architecture 147
CICS-CICS communication 17
CICS-IMS communication 105
CICS-to-CICS communication
application programming (LUTYPE6.1) 105
application programming (MRO) 53, 65
CICS-to-IMS communication 105
application programming (LUTYPE6.1) 113
CICS-to-IMS sessions
session allocation 110
client/server model 8
command sequences
APPC basic conversations 91
commands
APPC basic conversations 91
APPC mapped conversations 40
CICS-to-IMS sessions 112
LUTYPE6.1 conversations 112
MRO conversations 63
MRO mapped conversations 63
committing changes
to recoverable resources 119
CONFIRM option
GDS SEND command 76, 81
SEND command (APPC mapped) 33
CONNECT PROCESS command
APPC basic conversations 72
APPC mapped conversations 24, 40
PARTNER option 24
PIPLENGTH option 25
PIPLIST option 25
CONVDATA fields 86
conversation data block (CDB) 86
layout 87
conversation state 5
conversations
definition 5
CONVERSE command
APPC mapped conversations 31, 32, 38, 40
LUTYPE6.1 conversations 107, 109, 112
MRO conversations 60, 61, 63
CONVID option 55, 71
APPC basic conversations 102
APPC mapped conversations 25, 27
GDS CONNECT PROCESS command 72
LUTYPE6.1 conversations 105, 106
mandatory 102
WAIT command 27

D

data integrity 8
data streams, generalized
GDS for APPC 75
deferred transmission 36, 40, 76, 84, 92
APPC mapped conversations 27
MRO sessions 173
designing for recovery 12
DFHCDBLK copybook 87
distributed process 7
distributed program link 3
distributed transaction processing (DTP) 3
application programming 53, 65, 105, 113
CICS-to-CICS (LUTYPE6.1) 113
CICS-to-CICS (MRO) 53, 65
CICS-to-IMS (LUTYPE6.1) 105
distributed unit of work 8
DTP (distributed transaction processing) 3
DTP command 6

E

EIB fields 41, 65, 113, 121
 EIBATT
  LUTYPE6.1 conversations 109
  MRO conversations 55, 61
 EIBCOMPL
  APPC mapped conversations 38
  LUTYPE6.1 conversations 109
  MRO conversations 61
 EIBCONF
  APPC mapped conversations 34, 38
 EIBEOC
  APPC mapped conversations 38
 EIBERR 120
  APPC mapped conversations 27, 29, 32, 34, 37
 EIBERRCD
  APPC mapped conversations 27, 29, 32, 34, 37
 EIBFMH
  LUTYPE6.1 conversations 109
  MRO conversations 62
 EIBFREE 121
  APPC mapped conversations 27, 32, 34, 37
  LUTYPE6.1 conversations 108
  MRO conversations 61
 EIBNODAT
  APPC mapped conversations 29, 38
 EIBRCODE
  APPC mapped conversations 37
  LUTYPE6.1 conversations 105, 107, 111
  MRO conversations 61
 EIBRECV
  APPC mapped conversations 34, 38
  LUTYPE6.1 conversations 109
  MRO conversations 62
 EIBRLDBK 120, 139, 141
 EIBRSRCE
  APPC mapped conversations 23
  LUTYPE6.1 conversations 105, 111
 EIBSIG 115
  APPC mapped conversations 32, 37
  LUTYPE6.1 conversations 107, 167
 EIBSYNC 119, 140
  APPC mapped conversations 38
  LUTYPE6.1 conversations 108
  MRO conversations 61
 EIBSYNRB 120, 141
  APPC mapped conversations 37, 38
  MRO conversations 61
 MRO conversations 58, 61
EIBATT flag
 LUTYPE6.1 conversations 109
 MRO conversations 55, 61
EIBCOMPL flag
 APPC mapped conversations 38
 LUTYPE6.1 conversations 109
 MRO conversations 61
EIBCONF flag
 APPC mapped conversations 34, 38
EIBEOC flag
 APPC mapped conversations 38
EIBERR flag 120
EIBERR flag *(continued)*
 APPC mapped conversations 27, 29
EIBERRCD field
 APPC mapped conversations 27, 29, 32, 34, 37
EIBFMH flag
 LUTYPE6.1 conversations 109
 MRO conversations 62
EIBFREE flag 121
 APPC mapped conversations 27, 32, 34, 37
 LUTYPE6.1 conversations 108
 MRO conversations 61
EIBNODAT flag
 APPC mapped conversations 29, 38
EIBRCODE field
 APPC mapped conversations 37
 LUTYPE6.1 conversations 105, 107, 108
 MRO conversations 61
EIBRECV flag
 APPC mapped conversations 34, 38
 LUTYPE6.1 conversations 109
 MRO conversations 62
EIBRLDBK flag 120, 139, 141
EIBRSRCE field 53
 APPC mapped conversations 23
 LUTYPE6.1 conversations 105, 111
EIBSIG flag 115
 APPC mapped conversations 32, 37
 LUTYPE6.1 conversations 107, 167
EIBSYNC flag 119, 140
 APPC mapped conversations 38
 LUTYPE6.1 conversations 108
 MRO conversations 61
EIBSYNRB flag 120, 141
 APPC mapped conversations 37, 38
 MRO conversations 61
ending a conversation
 APPC basic sessions 83
 APPC mapped session 35
 LUTYPE6.1 sessions 108
 MRO session 60
EXTRACT ATTACH command
 LUTYPE6.1 conversations 106, 109, 112
 MRO conversations 55, 61, 63
EXTRACT ATTRIBUTES STATE command 6, 14
EXTRACT PROCESS command
 APPC basic conversations 73, 91
 APPC mapped conversations 25, 26, 40

F

failures
 back-end transaction 26, 56, 74, 106
 conversation 23, 77
 intersystem session 5, 13
 notification of 27, 74
FMH (function management header) 106, 175
 concatenated 62, 109
FREE command
 APPC basic conversations 83, 84, 92
 APPC mapped conversations 35, 36, 40
 CICS-to-IMS sessions 112

FREE command *(continued)*
LUTYPE6.1 conversations 108, 109, 112
MRO conversations 60, 61, 63
front-end transaction 6, 14
APPC basic conversations 71
APPC mapped conversations 23
LUTYPE6.1 conversations 105, 165
LUTYPE6.1 sessions (CICS-to-IMS) 110
MRO conversations 53
function management header (FMH) 106, 175
concatenated 62, 109
function shipping 3

G

GDS ALLOCATE command 86, 91
APPC basic conversations 71
PARTNER option 72
GDS ASSIGN command 73
GDS CONNECT PROCESS command 72, 91
PARTNER option 72
PIPLENGTH option 73
PIPLIST option 73
GDS EXTRACT PROCESS command 73, 91
GDS FREE command 83, 92
GDS ISSUE ABEND command 80
GDS ISSUE CONFIRMATION command 82
GDS ISSUE ERROR command 81
GDS ISSUE PREPARE command 120
GDS ISSUE SIGNAL command 80
GDS RECEIVE command 77, 88
BUFFER option 79
LLID option 78
GDS SEND command 75
GDS WAIT command 72, 76
generalized data stream (GDS)
GDS for APPC 75

H

header, function management 106, 109, 175

I

IMS 103, 105
integrity of data 8
INVITE option
GDS SEND command 76
SEND command (APPC mapped) 28
SEND command (LUTYPE6.1) 106
SEND command (MRO) 57
ISSUE ABEND command
APPC basic conversations 80
APPC mapped conversations 32
ISSUE CONFIRMATION command
APPC basic conversations 82
APPC mapped conversations 33
ISSUE ERROR command
APPC basic conversations 81
APPC mapped conversations 32
ISSUE PREPARE command 120
ISSUE SIGNAL command
APPC basic conversations 80
LUTYPE6.1 sessions (CICS-to-IMS) 112

L

LAST option
APPC sessions
with syncpointing 174
MRO sessions 174
with syncpointing 174
LLID option
GDS RECEIVE command 78
LUTYPE6.1 conversations
ALLOCATE command 105, 109, 112, 170
attaching partner transactions 105
back-end transaction 105, 165
BUILD ATTACH command 106
CICS-to-CICS application programming 113
CONVERSE command 107, 109, 112
CONVID option 105, 106
ending one 108
EXTRACT ATTACH command 106, 109, 112
FREE command 108, 109, 112
front-end transaction 105, 165
RECEIVE command 112
SEND command 106, 112

M

mapping to APPC architecture 147
basic (unmapped) conversations 148
mapped conversations 155
migration
LUTYPE6.1 programs on APPC links 165
migration mode 165
model
client/server 8
peer-to peer 8
MRO conversations
ALLOCATE command 53, 62, 63, 67
ASSIGN command 56
attaching partner transactions 54
back-end transaction 53
BUILD ATTACH command 54, 63
CONVERSE command 60, 61, 63
ending one 60
EXTRACT ATTACH command 55, 61, 63
FREE command 60, 61, 63
front-end transaction 53
RECEIVE command 61
Multi-Region Operation (MRO)
CICS-to-CICS application programming 53, 65, 105

N

NOQUEUE option
ALLOCATE command
LUTYPE6.1 sessions (CICS-to-IMS) 110

P

PARTNER option
ALLOCATE command 23
CONNECT PROCESS command 24
GDS ALLOCATE command 72
GDS CONNECT PROCESS command 72
peer-to-peer model 8
persistent session support, VTAM 18, 23, 71
PIP data
format of 25, 73
PIPLENGTH option
CONNECT PROCESS command 25
GDS CONNECT PROCESS command 73
PIPLIST option
CONNECT PROCESS command 25
GDS CONNECT PROCESS command 73
preparing a partner for syncpoint 120
principal facility 7
PROFILE option
ALLOCATE command
LUTYPE6.1 sessions (CICS-to-IMS) 110
ALLOCATE command (MRO) 53, 62
program development
APPC basic conversations 71
APPC mapped conversations 23
LUTYPE6.1 conversations 105
MRO conversations 53
programming
APPC basic conversations 71
APPC mapped conversations 23
LUTYPE6.1 conversations 105
MRO conversations 53
PSDINT, system initialization parameter 18

R

RECEIVE command
APPC basic conversations 77, 88
APPC mapped conversations 38
LUTYPE6.1 conversations 112
MRO conversations 61
recoverable resources 8
canceling changes to 8, 120
committing changes to 8, 119
RETCODE values 85
rollback 8
RTIMOUT attribute
PROFILE definition 36, 38

S

SEND command
APPC basic conversations 75
APPC mapped conversations 27, 28, 56
CONFIRM option
APPC mapped conversations 33
LUTYPE6.1 conversations 106, 112
session allocation
APPC basic conversations 71
LUTYPE6.1 conversations 110
SESSION option
ALLOCATE command (LUTYPE6.1) 110
sessions
allocating under ATI 23, 71
what they are 6
SNA (Systems Network Architecture) 9
starting a conversation
APPC basic 71
state of a conversation 5
STATE option 6, 14
GDS ALLOCATE command
APPC basic conversations 71
state tables
APPC basic conversations
sync level 0 94
sync level 1 96
sync level 2 98
APPC mapped conversations
sync level 0 42
sync level 1 44
sync level 2 46
LUTYPE6.1 conversations 114
migration mode 168
MRO conversations 66
state transitions
APPC basic conversations 93
state variable 6
sync level 9
synchronization 8
levels of 9
syncpoint 8
preparing a partner for 120
SYNCPOINT command 119
SYNCPOINT ROLLBACK command 120
APPC basic conversations 91
SYSID option
ALLOCATE command
LUTYPE6.1 sessions (CICS-to-IMS) 110
GDS ALLOCATE command
APPC basic conversations 71
system initialization parameters
PSDINT 18
Systems Network Architecture (SNA) 9

T

termination, abnormal
APPC basic conversations 80, 84
APPC mapped conversations 32, 34, 36
LUTYPE6.1 conversations 108
MRO conversations 60
testing the conversation state 50
transaction routing 3
transactions
back-end 6, 14
front-end 6, 14

U

unit of work (UOW) 8
UOW (unit of work) 8

V

VTAM
   persistent session support 18, 23, 71

W

WAIT command
   APPC basic conversations 72, 76
   APPC mapped conversations 24, 27
   LUTYPE6.1 conversations 106, 112
WAIT option
   GDS SEND command 76
   LUTYPE6.1 conversations 108
   SEND command
      MRO conversations 60
   SEND command (LUTYPE6.1) 106
   SEND command (MRO) 57
WAIT option (APPC mapped)
   SEND command 27
WAIT SIGNAL command 107

Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply in the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore this statement may not apply to you.

This publication could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM United Kingdom Laboratories, MP151, Hursley Park, Winchester, Hampshire, England, SO21 2JN. Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Programming License Agreement, or any equivalent agreement between us.

Programming Interface Information

This book is intended to help you understand how to program CICS systems to communicate with each other and with other systems. This book documents General-use Programming Interface and Associated Guidance Information provided by CICS. General-use programming interfaces allow the customer to write programs that obtain the services of CICS.

Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States, or other countries, or both:

ACF/VTAM	ES/9000	MVS/ESA
BookManager	IBM	OS/390
C/370	IBMLink	PR/SM
CICS	IMS	VTAM
CICS/ESA	IMS/ESA	

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

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

Sending your comments to IBM

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

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

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

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

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

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

- By mail, to this address:

  User Technologies Department (MP095)
  IBM United Kingdom Laboratories
  Hursley Park
  WINCHESTER,
  Hampshire
  SO21 2JN
  United Kingdom
- By fax:
  - From outside the U.K., after your international access code use 44–1962–816151
  - From within the U.K., use 01962–816151
- Electronically, use the appropriate network ID:
  - IBM Mail Exchange: GBIBM2Q9 at IBMMAIL
  - IBMLink™: HURSLEY(IDRCF)
  - Internet: idrcf@hursley.ibm.com

Whichever you use, ensure that you include:

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

Program Number: 5697-E93

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SC34-5998-00

Spine information:

CICS TS for z/OS

CICS Distributed Transaction Programming Guide

Version 2
Release 2