CICS Transaction Server for z/OS
5.6

Intercommunication Guide

IBM

Note

Before using this information and the product it supports, read the information in Product Legal Notices.

This edition applies to the IBM® CICS® Transaction Server for z/OS®, Version 5 Release 6 (product number 5655-Y305655-BTA) and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

About this PDF

This PDF describes how to connect CICS systems using multiregion operation (MRO), intersystem communication over SNA (ISC over SNA), or IP interconnectivity (IPIC). It is aimed at system programmers who are responsible for planning and implementing these types of connection. Other PDFs, listed below, contain details for other types of CICS connectivity.

This PDF covers:

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

Information about accessing CICS programs and transactions for specific areas of CICS is in the following PDFs:

- From the internet is in the *Internet Guide* .
- From other non-CICS environments is in the *External Interfaces Guide* and *Using EXCI with CICS*.
- With BTS is in *Business Transaction Services*
- With FEPI is in the *Front End Programming Interface User's Guide*.
- DTP is in the *Distributed Transaction Programming Guide*.

For details of the terms and notation used in this book, see Conventions and terminology used in CICS documentation in IBM Documentation.

Date of this PDF

This PDF was created on 2024-04-22 (Year-Month-Date).

Chapter 1. CICS intercommunication

CICS is often used as a single system with associated data resources and a network of terminals. However, CICS can also be used in a multiple-system environment, in which it can communicate with other systems that have similar communication facilities. This sort of communication is called *CICS intercommunication*.

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

For information about accessing CICS programs and transactions from the Internet, see Internet, TCP/IP, and HTTP concepts. For information about accessing CICS programs and transactions from other non-CICS environments, see Overview of CICS external interfaces.

This section contains the following topics:

- "Intercommunication methods" on page 1
- "Intercommunication facilities" on page 4
- "Using CICS intercommunication" on page 2.

Intercommunication methods

CICS can communicate with other systems that are in the same operating system or sysplex using multiregion operation (MRO). To communicate with other CICS or non-CICS systems that are not in the same z/OS image or sysplex, CICS connects using either a TCP/IP (IPIC) or SNA (ISC over SNA) protocol.

Communication between systems

For communication between CICS and non-CICS systems, or between CICS systems that are not in the same operating system or z/OS sysplex, you usually require a network access method to provide the necessary communication protocols.

CICS TS for z/OS, Version 5.6 supports two such *intercommunication facilities*:

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

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

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

Multiregion operation

For CICS-to-CICS communication, CICS provides an **interregion communication** facility that does not require the use of a network access method such as ACF/SNA or TCP/IP.

This form of communication is called **multiregion operation** (MRO). MRO can be used between CICS regions that reside:

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

Note: The external CICS interface (EXCI) uses a specialized form of MRO link to support communication between MVS batch programs and CICS.

Using CICS intercommunication

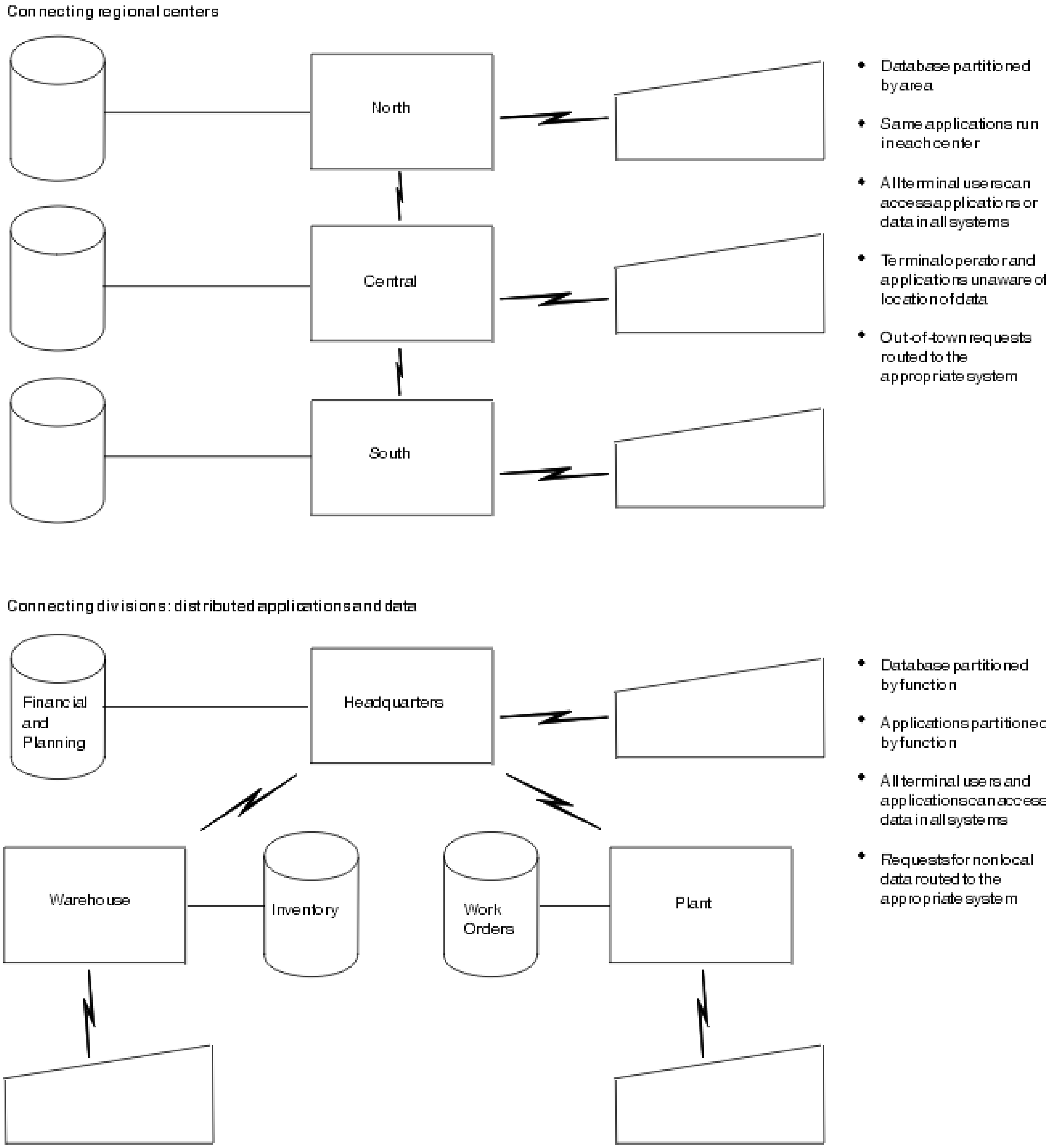
The CICS intercommunication facilities allow you to implement many different types of distributed transaction processing. Some examples of typical applications are explained.

Multiregion operation allows two CICS regions to share selected system resources, and to present a “single-system” view to terminal operators. At the same time, each region can run independently of the other, and can be protected against errors in other regions. Various possible applications of MRO are described in “Multiregion operation” on page 30.

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

IPIC connections allow you to use a TCP/IP network for intercommunication between systems. IPIC provides similar capabilities and qualities of service to those provided by ISC over SNA.

Figure 1 on page 3 shows some typical possibilities.



- Database partitioned by area
- Same applications run in each center
- All terminal users can access applications or data in all systems
- Terminal operator and applications unaware of location of data
- Out-of-town requests routed to the appropriate system

- Database partitioned by function
- Applications partitioned by function
- All terminal users and applications can access data in all systems
- Requests for nonlocal data routed to the appropriate system

Figure 1. Examples of distributed resources (Part 1)

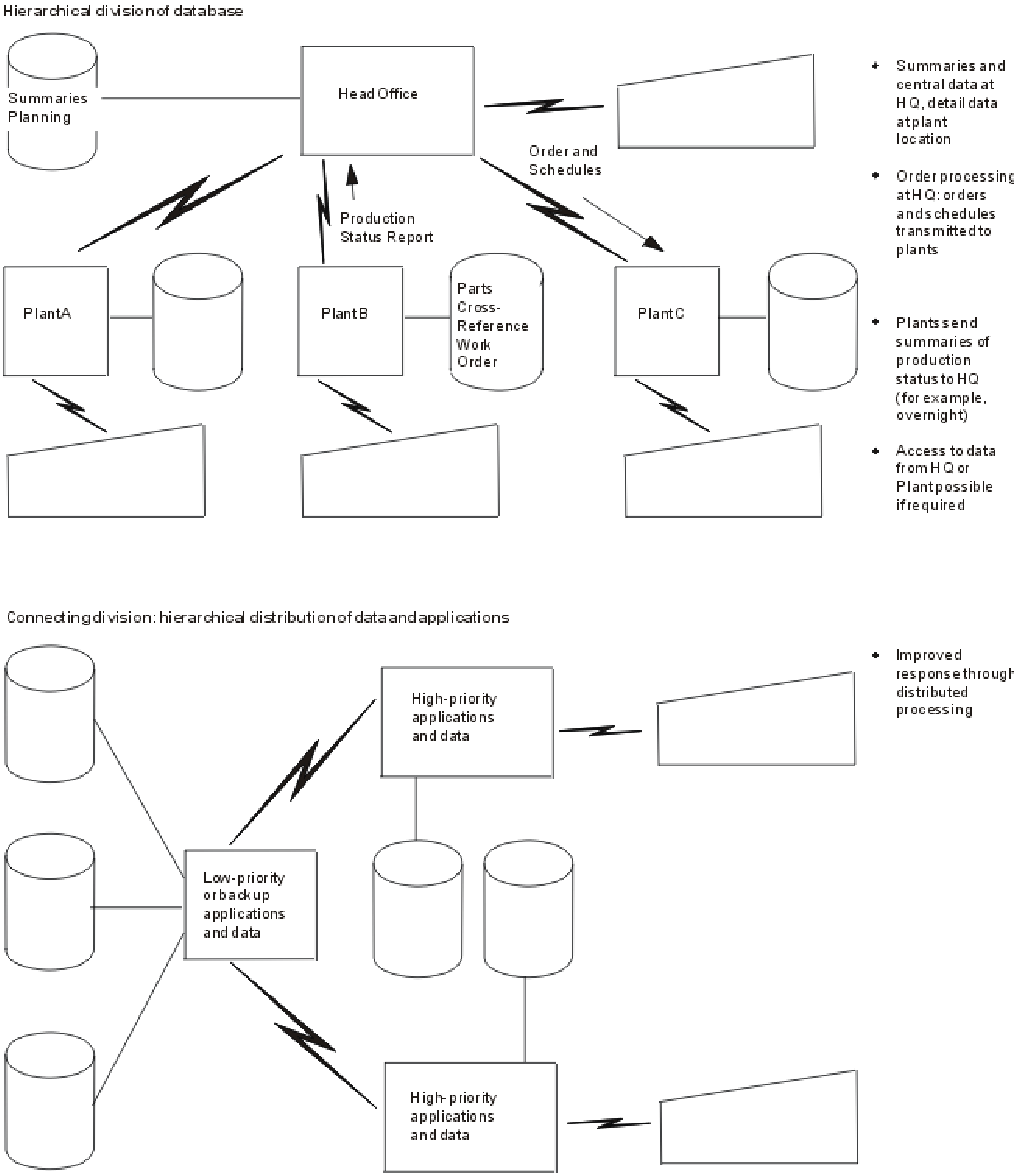

Hierarchical division of database
Summaries Planning
Head Office
Order and Schedules
Production Status Report
Plant A
Plant B
Parts Cross-Reference Work Order
Plant C
Summaries and central data at HQ, detail data at plant location
Order processing at HQ: orders and schedules transmitted to plants
Plants send summaries of production status to HQ (for example, overnight)
Access to data from HQ or Plant possible if required
Connecting division: hierarchical distribution of data and applications
High-priority applications and data
Low-priority or backup applications and data
High-priority applications and data
Improved response through distributed processing

Figure 2. Examples of distributed resources (Part 2)

Intercommunication facilities

In a multiple-system environment, each participating system can have its own local terminals and databases, and can run its local application programs independently of other systems in the network.

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

CICS provides the following types of facilities for communicating with other CICS, IMS, or other systems:

- Function shipping

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

For more information about the intercommunication facilities that support access to CICS programs and transactions from non-CICS environments, see Interfaces to CICS transactions and programs and Internet, TCP/IP, and HTTP concepts .

These communication facilities are not all available for all forms of intercommunication. The circumstances under which they can be used are shown in Table 1 on page 5.

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

	IRC Interregion communication	**Intersystem communication over TCP/IP**		**Intersystem communication over SNA (using ACF/ z/OS Communications Server)**			
	MRO	**IPIC**		**LUTYPE6.2 (APPC)**		**LUTYPE6.1**	
Facility	**CICS**	**CICS**	**non-CICS (for example, CICS TG)**	**CICS**	**non-CICS (for example, CICS TG)**	**CICS**	**IMS**
Function Shipping	Yes	Yes	No	Yes	No	Yes	No
Asynchronous Processing	Yes	Yes	No	Yes	No	Yes	Yes
Transaction Routing	Yes	Yes	Yes	Yes	No	No	No
Distributed program link	Yes	Yes	Yes	Yes	No	No	No
Distributed transaction processing	Yes	No	No	Yes	Yes	Yes	Yes

Function shipping

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

The following remote resources can be accessed using function shipping:

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

Application programs that access remote resources can be designed and coded as if the resources were owned by the system in which the transaction is to run. During execution, CICS ships the request to the appropriate system.

Function shipping is supported between CICS systems connected by IPIC, ISC over SNA, or MRO links. IPIC only supports function shipping of file control, transient data, and temporary storage requests between CICS TS 4.2 or later regions.

Asynchronous processing

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

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

Asynchronous processing is supported between CICS systems connected by MRO, or ISC over SNA links. IPIC supports asynchronous processing of **EXEC CICS START**, **START CHANNEL**, and **CANCEL** commands between CICS regions..

Transaction routing

Transaction routing allows a transaction and an associated terminal to be owned by different CICS systems.

Transaction routing can take the following forms:

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

Transaction routing is supported between CICS systems connected by IPIC, MRO, or ISC over SNA links. IPIC supports transaction routing of 3270 terminals between CICS TS 4.1 or later regions, where the terminal-owning region (TOR) is uniquely identified by an APPLID.

Distributed program link (DPL)

CICS distributed program link enables a CICS program (the client program) to call another CICS program (the server program) in a remote CICS region. IPIC supports DPL between CICS systems and between CICS TS and TXSeries®.

Why use DPL

Here are some of the reasons you might want to design your application to use DPL:

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

Distributed transaction processing (DTP)

The technique of distributing the functions of a transaction over several transaction programs within a network is called **distributed transaction processing (DTP)**. DTP allows a CICS transaction to communicate with a transaction running in another system. The transactions are designed and coded specifically to communicate with each other, and thereby to use the intersystem link with maximum efficiency.

The communication in DTP is, from the CICS point of view, **synchronous**, which means that it occurs during a single invocation of the CICS transaction and that requests and replies between two transactions can be directly associated. This contrasts with the asynchronous processing described previously.

DTP is supported between CICS systems connected by MRO, or ISC over SNA links.

Transaction tracking

Transaction tracking shows the origin of, and relationships between, tasks in an application as they flow across CICS regions in a CICSplex. Transaction tracking information is written out to SMF in a task's monitoring data and can help you with auditing and problem determination. Functions are provided to locate specific tasks based on information in the point of origin to find interrelated hung tasks, and to identify work initiated by non-CICS adapters (such as IBM MQ).

Transaction tracking provides tighter integration between other products, such as IBM MQ, and an extension of the scope of transaction tracking to other interfaces, including WebSphere® Optimized Local Adapter and CICS sockets. The IBM MQ task-related user exit (TRUE) supports transaction tracking.

Transaction tracking has the following features:

Association data

Association data is a set of information that describes the environment in which user tasks run and the way that user tasks are attached in a region. Association data is built during task attach processing and represents context information specific to the task itself; for example, the task ID, the user ID relating to the task, and the principal facility of the task. Association data is propagated across IPIC and MRO to provide a complete story across the CICSplex for all user tasks, including CICS transactions started by a user (for example, CEMT) or running on behalf of a user-initiated transaction (for example, CSMI). For more information about the components of association data, see Association data.

Point of origin

Every user task in a CICSplex has a point of origin, such as a web service request or an IBM MQ message. When work first enters the CICSplex, details of its point of origin is placed into task context information called *origin data* (part of its task association data) for the first task that is created to process it. This origin data flows with the work as it moves around the CICSplex. Transaction tracking tracks the point of origin of a transaction by associating an initial user task with other tasks that are created from it. The created tasks carry information about the initial user task as origin data.

Transaction group

A transaction group associates transactions. The transactions all contain the same unique identifier of the originating transaction in the TRNGRPID. You use the TRNGRPID to track where transactions are created when they do not share a unit of work.

Adapter tracking

Adapter tracking tracks tasks that are created by non-CICS transports (for example, adapters that connect to other software applications such as IBM MQ), which can participate in transaction tracking. The adapters can add unique task metadata, describing the origin, into the propagated context of each transaction they initiate. This adapter data is carried in the origin data section of the association data and can be used to track the transactions started by the adapter.

Examples of adapter data added to a task's origin data is when tasks start as a result of requests coming into CICS from z/OS Connect over the IPIC protocol. z/OS Connect passes adapter data, which is added to the origin data. This feature requires z/OS Connect Enterprise Edition 3.0.30.0 or later. Another example is when the CICS-MQ trigger monitor or CICS-MQ bridge initiate tasks as a result of the arrival of MQ messages. The CICS-MQ adapter will append adapter data giving information about the MQ queue involved.

Adapter data is written as part of the origin data into the CICS 110 SMF monitoring record. If the task accesses Db2®, the CICS-Db2 attach will detect the presence of adapter data and, if ACCOUNTREC(UOW) or ACCOUNTREC(TASK) is set on the DB2CONN or DB2ENTRY definition, will pass the data to Db2.

The adapter ID, preceded by an 28-character eyecatcher, is passed as **appl-longname** as follows:

```
CICS_ORIGIN_DATA:ADAPTER_ID:adapter_id
```

The adapter data, preceded by an 36-character eyecatcher, is passed as an **accounting-string** as follows:

```
CICS_ORIGIN_DATA:ADAPTER_DATA_1_2_3:adapter_data
```

Db2 will write the data in its SMF accounting records and the data is also available online through the Db2 special registers CURRENT CLIENT_APPLNAME and CURRENT CLIENT_ACCTNG.

Note:

- This capability requires CICS TS 5.6 with APAR PH30252 and Db2 12 with APAR PH31447 or higher.
- You can disable the passing of adapter origin data to Db2 by specifying the following feature toggle:

  ```
  com.ibm.cics.db2.origindata=false
  ```

Association data

Association data is a set of information that describes the environment in which user tasks run and the way that user tasks are attached in a region. User tasks are tasks that are associated with user-defined transactions or with transactions that are supplied by CICS. CEMT is an example of a user-initiated task that is typically started by an operator, and CSMI is an example of a task that is started by the system on behalf of a user-initiated transaction.

Association data is built during task attach processing and represents context information specific to the task itself; for example, the task ID, the user ID relating to the task, and the principal facility of the task. Association data can also include details about the origin of the task and the way it was started.

You can use CICS Explorer®, the CICSPlex SM WUI, or the INQUIRE ASSOCIATION and INQUIRE ASSOCIATION LIST commands to view association data. You can use the CICS Performance Analyzer (CICS PA) and the sample monitoring data print program, DFH$MOLS to report on association data. See "Viewing association data" on page 10 for examples of the data.

The following data components support transaction tracking:

Adapter data
Adapter data is a part of the origin data section of association data and can be defined and provided by an adapter from other software that introduces work into CICS. This data can include, for example, data to identify which adapter started the task. The adapter data can then be used to track the transactions started by the adapter. For information about using adapter data for tracking transactions, see Adapter tracking sample task-related user exit program (DFH$APDT).

ApplData
Association data uses socket application data (ApplData) for the socket that received the request to start the task. You can use the ApplData to correlate TCP/IP connections with the CICS regions and transactions that are using them. In TCP/IP, the ApplData information is available on the Netstat ALL/-A, ALLConn/-a, and COnn/-c reports, and can be searched with the APPLD/-G filter. See z/OS Communications Server: IP System Administrator's Commands for additional information about using ApplData with Netstat. The ApplData information is available in the SMF 119 TCP Connection Termination record. See z/OS Communications Server: IP Configuration Reference for additional information. The ApplData information is also available through the Network Management Interface. See z/OS Communications Server: IP Programmer's Guide and Reference for more information.

Application context data

Application context data is provided for CICS applications that are deployed on a platform. Application context data identifies the application, operation, application version, and the platform in which the application is running. To generate application context data, the application must have a declared set of application entry points, which define how other applications and users enter the application. CICS then adds application context data to each task at the point the application is entered. For information about using application context data, see Application context.

Origin data

Origin data is a section of association data that describes where the task was started (the point of origin). Origin data is created by a user task that is started when an external request arrives at a CICSplex. For information about origin data, see Origin data characteristics.

Previous hop data

Previous hop data is a section of association data that describes the remote sender of the request so that the request can be tracked back into the previous system. For information about previous hop data, see Previous hop data characteristics.

Previous transaction data

Previous transaction data is a section of association data that describes the local or parent task of a request to attach a task by a RUN TRANSID or START TRANSID command (when a new point of origin is not created). For information about previous hop data, see "Previous transaction data characteristics" on page 21.

Task context data

Task context data is a section of association data that provides information about the specific context of the user task that is being referenced.

User correlation data

User correlation data is a part of the origin data section of association data and is added by the XAPADMGR global user exit program. You can use the XAPADMGR exit to add user information at the point of origin of the interrelated transactions. For further information about using user information for tracking transactions, see Application association data exit in the AP domain (XAPADMGR).

Figure 3 on page 10 represents how association data passes between tasks. Both previous hop counts and previous transaction counts are represented, with previous hop counts flowing horizontally between CICS regions, and previous transaction counts flowing vertically down within a CICS region. In this example, association data moves between regions using a combination of distributed program link (DPL) and function ship (FS).

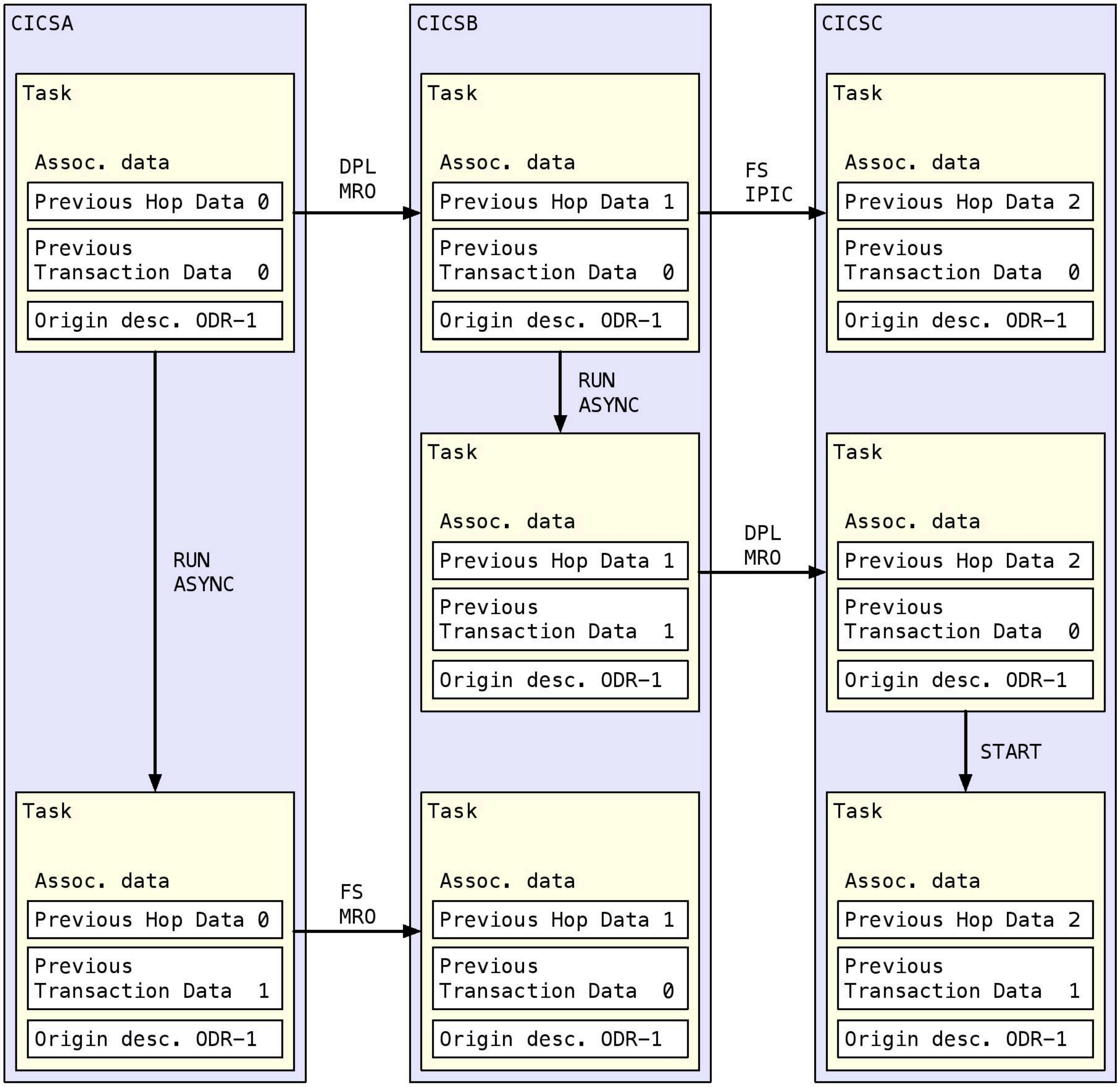


Figure 3. Visualizing task relationships

Viewing association data

You can use CICS Explorer, the CICSPlex SM WUI, or the INQUIRE ASSOCIATION and INQUIRE ASSOCIATION LIST commands to view association data. You can use the CICS Performance Analyzer (CICS PA) and the sample monitoring data print program, DFH$MOLS, to report on association data.

Example: viewing association data with DFH$MOLS

When running DFH$MOLS against your SMF 110 records, your transaction tracking data might look something like this for a parent transaction called ASPA:

```
DFHTASK  P031     TRANNUM          0000042C                                        42
DFHCICS  C363     OTRAN            C1E2D7C1                                     ASPA

DFHCICS  C373     PHNTWKID         00000000 00000000
DFHCICS  C374     PHAPPLID         00000000 00000000
DFHCICS  P376     PHTRANNO         0000000C                                         0
DFHCICS  C377     PHTRAN           00000000
DFHCICS  A378     PHCOUNT          00000000                                         0
DFHCICS  P481     PTTRANNO         0000000C                                         0
DFHCICS  C482     PTTRAN           00000000
DFHCICS  A483     PTCOUNT          00000000                                         0
```

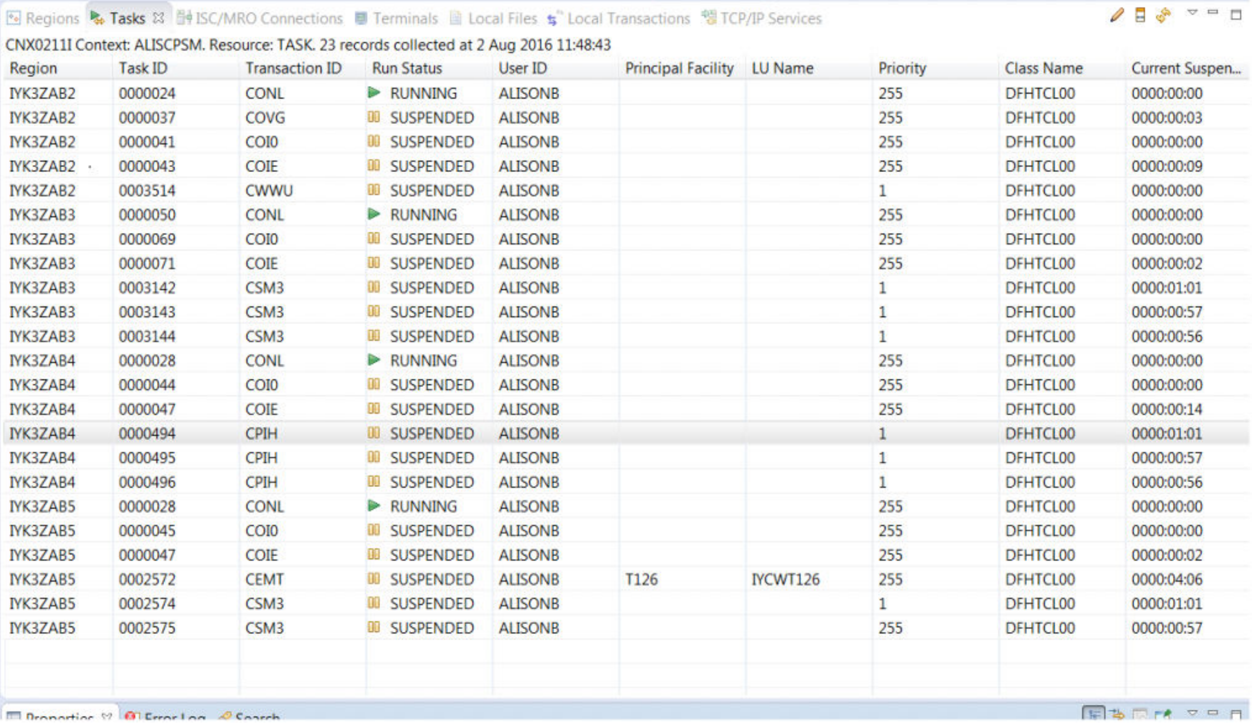
and like this for a child task called ASCH:

```
DFHTASK  P031     TRANNUM          0000043C                                        43
DFHCICS  C363     OTRAN            C1E2D7C1                                      ASPA

DFHCICS  C373     PHNTWKID         00000000 00000000
DFHCICS  C374     PHAPPLID         00000000 00000000
DFHCICS  P376     PHTRANNO         0000000C                                         0
DFHCICS  C377     PHTRAN           00000000
DFHCICS  A378     PHCOUNT          00000000                                         0

DFHCICS  T480     PTSTART          D169DEEC8300D482                        2016/09/28
09:04:29.011981
DFHCICS  P481     PTTRANNO         0000042C                                        42
DFHCICS  C482     PTTRAN           C1E2D7C1                                      ASPA
DFHCICS  A483     PTCOUNT          00000001                                         1
```

The child task ASCH has a PTTRAN value of ASPA, which identifies its parent, and PTCOUNT of 1, which indicates that it is one level deep within the CICS region (that is, it is a child task but *not* a grandchild). If task ASCH had children of its own, those children would have a PTTRAN value of ASCH, and a PTCOUNT of 2, which would identify those children as grandchildren of task ASPA.

Example: viewing association data with CICS Explorer

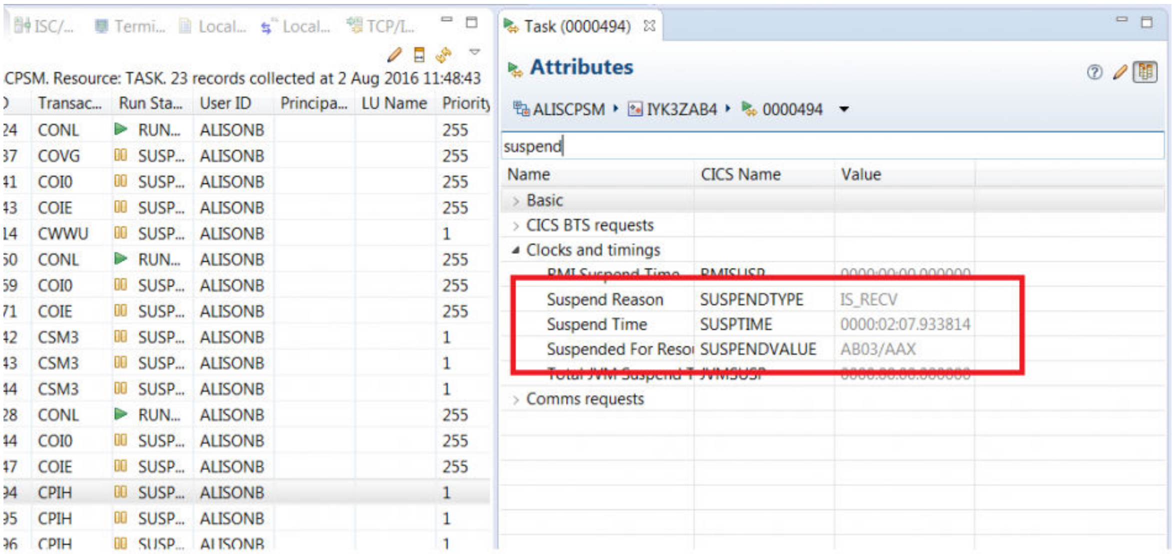
Assume that users suddenly report that their application is hanging. You know that the processing is by a web service request into CICS, and that this web service makes use of default pipeline processing, so you use the Tasks view in CICS Explorer to find all the instances of the CPIH pipeline transaction:



CNX0211I Context: ALISCPSM. Resource: TASK. 23 records collected at 2 Aug 2016 11:48:43

Region	Task ID	Transaction ID	Run Status	User ID	Principal Facility	LU Name	Priority	Class Name	Current Suspen...
IYK3ZAB2	0000024	CONL	▶ RUNNING	ALISONB			255	DFHTCL00	0000:00:00
IYK3ZAB2	0000037	COVG	⏸ SUSPENDED	ALISONB			255	DFHTCL00	0000:00:03
IYK3ZAB2	0000041	COI0	⏸ SUSPENDED	ALISONB			255	DFHTCL00	0000:00:00
IYK3ZAB2	0000043	COIE	⏸ SUSPENDED	ALISONB			255	DFHTCL00	0000:00:09
IYK3ZAB2	0003514	CWWU	⏸ SUSPENDED	ALISONB			1	DFHTCL00	0000:00:00
IYK3ZAB3	0000050	CONL	▶ RUNNING	ALISONB			255	DFHTCL00	0000:00:00
IYK3ZAB3	0000069	COI0	⏸ SUSPENDED	ALISONB			255	DFHTCL00	0000:00:00
IYK3ZAB3	0000071	COIE	⏸ SUSPENDED	ALISONB			255	DFHTCL00	0000:00:02
IYK3ZAB3	0003142	CSM3	⏸ SUSPENDED	ALISONB			1	DFHTCL00	0000:01:01
IYK3ZAB3	0003143	CSM3	⏸ SUSPENDED	ALISONB			1	DFHTCL00	0000:00:57
IYK3ZAB3	0003144	CSM3	⏸ SUSPENDED	ALISONB			1	DFHTCL00	0000:00:56
IYK3ZAB4	0000028	CONL	▶ RUNNING	ALISONB			255	DFHTCL00	0000:00:00
IYK3ZAB4	0000044	COI0	⏸ SUSPENDED	ALISONB			255	DFHTCL00	0000:00:00
IYK3ZAB4	0000047	COIE	⏸ SUSPENDED	ALISONB			255	DFHTCL00	0000:00:14
IYK3ZAB4	0000494	CPIH	⏸ SUSPENDED	ALISONB			1	DFHTCL00	0000:01:01
IYK3ZAB4	0000495	CPIH	⏸ SUSPENDED	ALISONB			1	DFHTCL00	0000:00:57
IYK3ZAB4	0000496	CPIH	⏸ SUSPENDED	ALISONB			1	DFHTCL00	0000:00:56
IYK3ZAB5	0000028	CONL	▶ RUNNING	ALISONB			255	DFHTCL00	0000:00:00
IYK3ZAB5	0000045	COI0	⏸ SUSPENDED	ALISONB			255	DFHTCL00	0000:00:00
IYK3ZAB5	0000047	COIE	⏸ SUSPENDED	ALISONB			255	DFHTCL00	0000:00:02
IYK3ZAB5	0002572	CEMT	⏸ SUSPENDED	ALISONB	T126	IYCWT126	255	DFHTCL00	0000:04:06
IYK3ZAB5	0002574	CSM3	⏸ SUSPENDED	ALISONB			1	DFHTCL00	0000:01:01
IYK3ZAB5	0002575	CSM3	⏸ SUSPENDED	ALISONB			255	DFHTCL00	0000:00:57

Double-click on the oldest instance of CPIH (task 494 on region IYK3ZAB4) to obtain its attributes and filter on suspend. This shows:



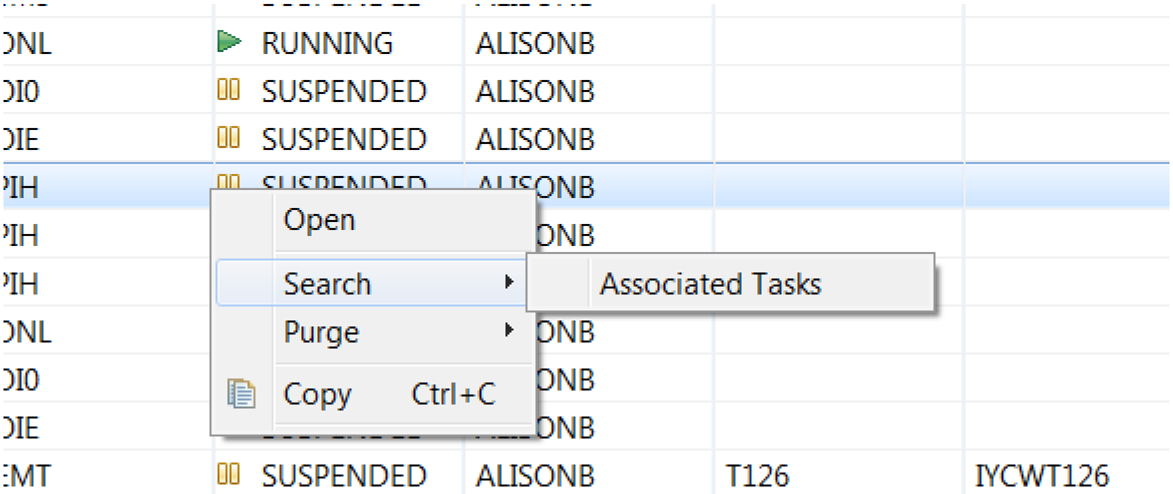
CPSM. Resource: TASK. 23 records collected at 2 Aug 2016 11:48:43

)	Transac...	Run Sta...	User ID	Principa...	LU Name	Priority
24	CONL	▶ RUN...	ALISONB			255
37	COVG	⏸ SUSP...	ALISONB			255
41	COI0	⏸ SUSP...	ALISONB			255
43	COIE	⏸ SUSP...	ALISONB			255
14	CWWU	⏸ SUSP...	ALISONB			1
50	CONL	▶ RUN...	ALISONB			255
59	COI0	⏸ SUSP...	ALISONB			255
71	COIE	⏸ SUSP...	ALISONB			255
42	CSM3	⏸ SUSP...	ALISONB			1
43	CSM3	⏸ SUSP...	ALISONB			1
44	CSM3	⏸ SUSP...	ALISONB			1
28	CONL	▶ RUN...	ALISONB			255
44	COI0	⏸ SUSP...	ALISONB			255
47	COIE	⏸ SUSP...	ALISONB			255
94	CPIH	⏸ SUSP...	ALISONB			1
95	CPIH	⏸ SUSP...	ALISONB			1
96	CPIH	⏸ SUSP...	ALISONB			1

suspend

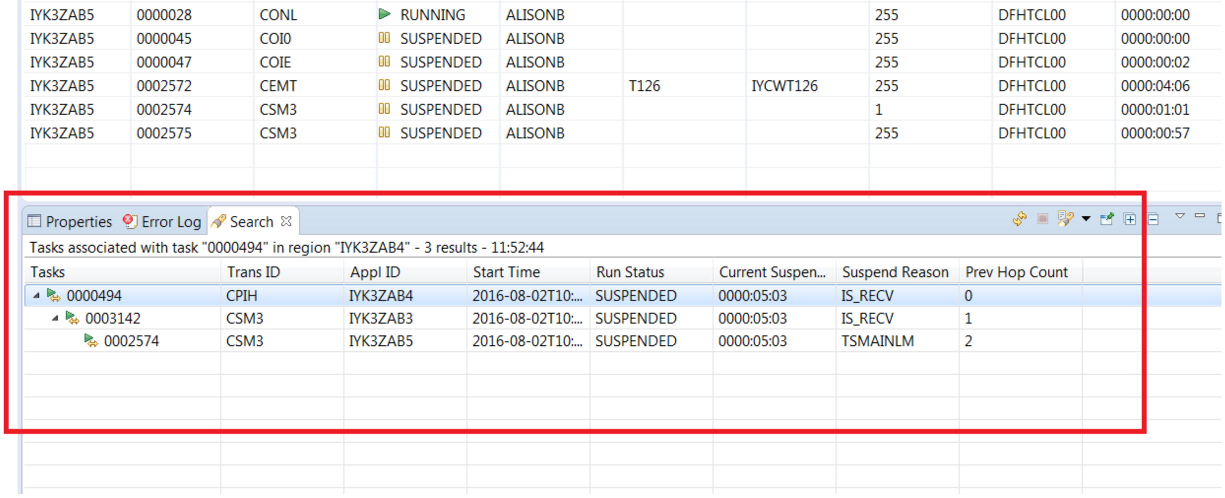
Name	CICS Name	Value
› Basic		
› CICS BTS requests		
◢ Clocks and timings		
RMI Suspend Time	RMISUSP	0000:00:00.000000
Suspend Reason	SUSPENDTYPE	IS_RECV
Suspend Time	SUSPTIME	0000:02:07.933814
Suspended For Reso	SUSPENDVALUE	AB03/AAX
Total JVM Suspend T	JVMSUSP	0000:00:00.000000
› Comms requests		

You can see that the work is suspended and waiting on a response over IPCONN AB03 which is connected to region IYK3ZAB3. Right-click this top instance of CPIH to search for associated tasks:



ONL	▶ RUNNING	ALISONB		
OI0	⏸ SUSPENDED	ALISONB		
OIE	⏸ SUSPENDED	ALISONB		
PIH	⏸ SUSPENDED	ALISONB		
PIH		ONB		
PIH				
ONL		ONB		
OI0		ONB		
OIE		ONB		
EMT	⏸ SUSPENDED	ALISONB	T126	IYCWT126

This shows which tasks are associated with this instance of CPIH in the Search results tab:



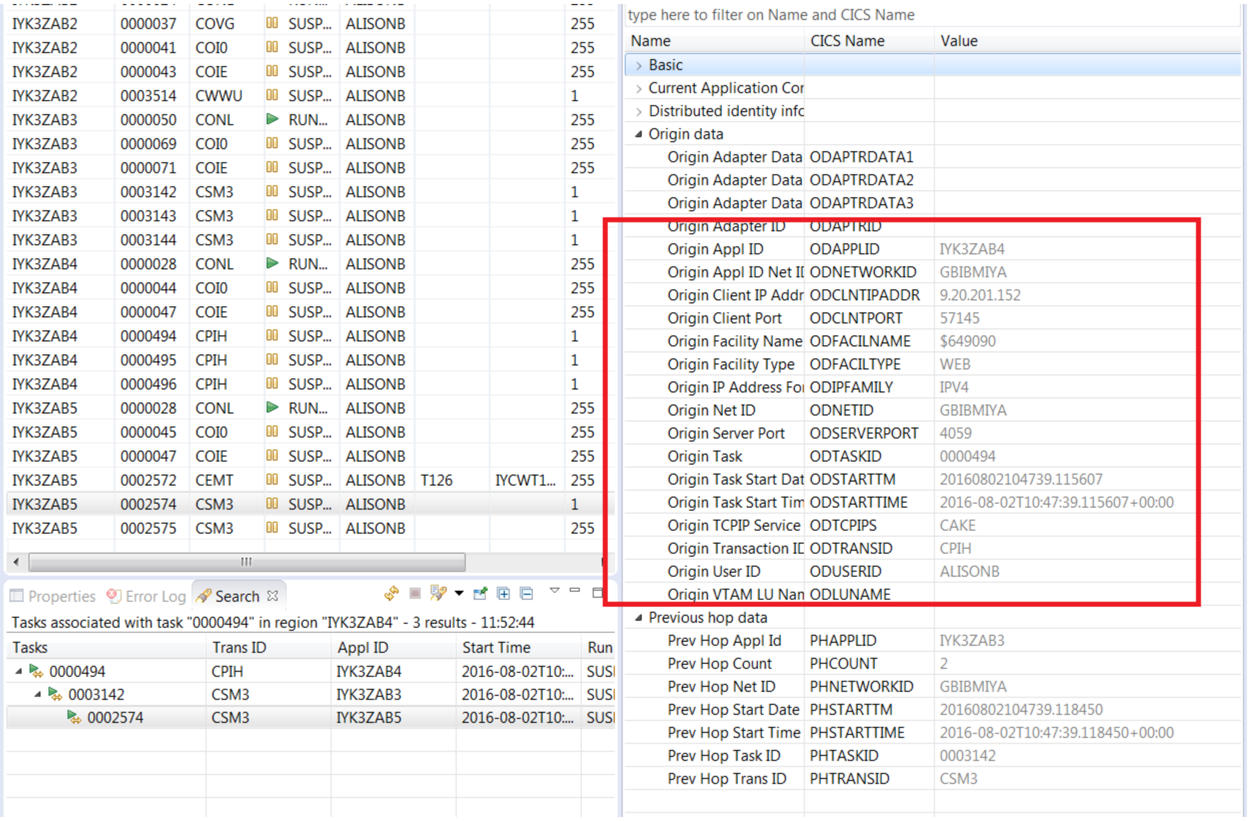
IYK3ZAB5	0000028	CONL	▶ RUNNING	ALISONB			255	DFHTCL00	0000:00:00
IYK3ZAB5	0000045	COI0	⏸ SUSPENDED	ALISONB			255	DFHTCL00	0000:00:00
IYK3ZAB5	0000047	COIE	⏸ SUSPENDED	ALISONB			255	DFHTCL00	0000:00:02
IYK3ZAB5	0002572	CEMT	⏸ SUSPENDED	ALISONB	T126	IYCWT126	255	DFHTCL00	0000:04:06
IYK3ZAB5	0002574	CSM3	⏸ SUSPENDED	ALISONB			1	DFHTCL00	0000:01:01
IYK3ZAB5	0002575	CSM3	⏸ SUSPENDED	ALISONB			255	DFHTCL00	0000:00:57

Tasks associated with task "0000494" in region "IYK3ZAB4" - 3 results - 11:52:44

Tasks	Trans ID	Appl ID	Start Time	Run Status	Current Suspen...	Suspend Reason	Prev Hop Count
◢ 0000494	CPIH	IYK3ZAB4	2016-08-02T10:...	SUSPENDED	0000:05:03	IS_RECV	0
◢ 0003142	CSM3	IYK3ZAB3	2016-08-02T10:...	SUSPENDED	0000:05:03	IS_RECV	1
0002574	CSM3	IYK3ZAB5	2016-08-02T10:...	SUSPENDED	0000:05:03	TSMAINLM	2

You know that the associated mirror task on IYK3ZAB3 is task 3142 which itself routed on to mirror task 2574 on region IYK3ZAB5. By default, the Search results tab shows the suspend reason for each task, so that you can see that it is suspended on TSMAINLM. As a result, you can check what is using all the temporary storage on IYK3ZAB5 and address this.

You can take a closer look at the origin data part of the association data. For the hanging mirror task, see this by right clicking on it in the Search results tab and selecting **Task Association** then **Open**:



IYK3ZAB2	0000037	COVG	SUSP...	ALISONB			255
IYK3ZAB2	0000041	COI0	SUSP...	ALISONB			255
IYK3ZAB2	0000043	COIE	SUSP...	ALISONB			255
IYK3ZAB2	0003514	CWWU	SUSP...	ALISONB			1
IYK3ZAB3	0000050	CONL	RUN...	ALISONB			255
IYK3ZAB3	0000069	COI0	SUSP...	ALISONB			255
IYK3ZAB3	0000071	COIE	SUSP...	ALISONB			255
IYK3ZAB3	0003142	CSM3	SUSP...	ALISONB			1
IYK3ZAB3	0003143	CSM3	SUSP...	ALISONB			1
IYK3ZAB3	0003144	CSM3	SUSP...	ALISONB			1
IYK3ZAB4	0000028	CONL	RUN...	ALISONB			255
IYK3ZAB4	0000044	COI0	SUSP...	ALISONB			255
IYK3ZAB4	0000047	COIE	SUSP...	ALISONB			255
IYK3ZAB4	0000494	CPIH	SUSP...	ALISONB			1
IYK3ZAB4	0000495	CPIH	SUSP...	ALISONB			1
IYK3ZAB4	0000496	CPIH	SUSP...	ALISONB			1
IYK3ZAB5	0000028	CONL	RUN...	ALISONB			255
IYK3ZAB5	0000045	COI0	SUSP...	ALISONB			255
IYK3ZAB5	0000047	COIE	SUSP...	ALISONB			255
IYK3ZAB5	0002572	CEMT	SUSP...	ALISONB	T126	IYCWT1...	255
IYK3ZAB5	0002574	CSM3	SUSP...	ALISONB			1
IYK3ZAB5	0002575	CSM3	SUSP...	ALISONB			255

Tasks associated with task "0000494" in region "IYK3ZAB4" - 3 results - 11:52:44

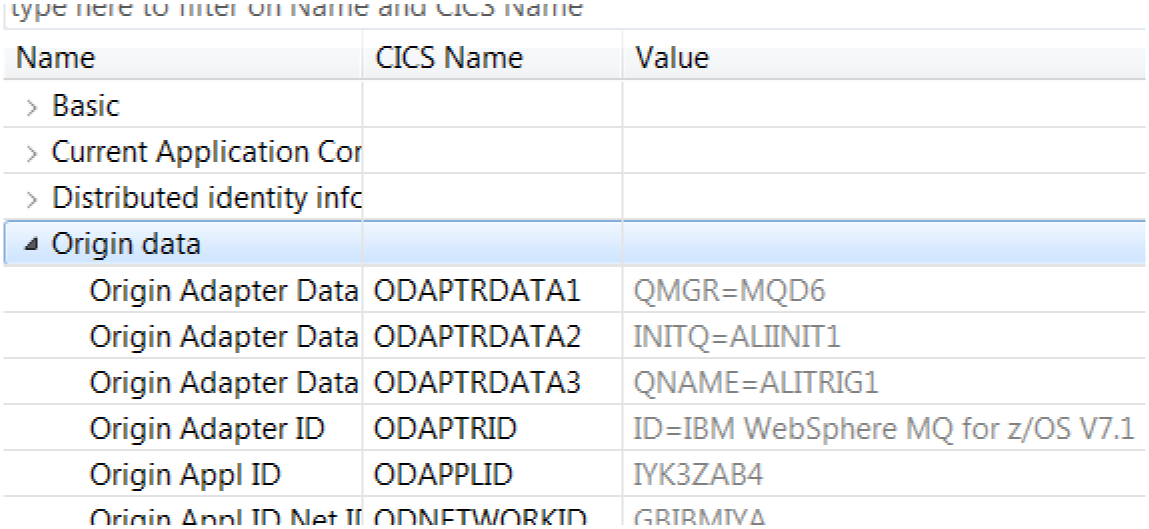
Tasks	Trans ID	Appl ID	Start Time	Run
0000494	CPIH	IYK3ZAB4	2016-08-02T10:...	SUSI
0003142	CSM3	IYK3ZAB3	2016-08-02T10:...	SUSI
0002574	CSM3	IYK3ZAB5	2016-08-02T10:...	SUSI

type here to filter on Name and CICS Name

Name	CICS Name	Value
› Basic		
› Current Application Cor		
› Distributed identity infc		
◢ Origin data		
Origin Adapter Data	ODAPTRDATA1	
Origin Adapter Data	ODAPTRDATA2	
Origin Adapter Data	ODAPTRDATA3	
Origin Adapter ID	ODAPTRID	
Origin Appl ID	ODAPPLID	IYK3ZAB4
Origin Appl ID Net II	ODNETWORKID	GBIBMIYA
Origin Client IP Addr	ODCLNTIPADDR	9.20.201.152
Origin Client Port	ODCLNTPORT	57145
Origin Facility Name	ODFACILNAME	$649090
Origin Facility Type	ODFACILTYPE	WEB
Origin IP Address Fo	ODIPFAMILY	IPV4
Origin Net ID	ODNETID	GBIBMIYA
Origin Server Port	ODSERVERPORT	4059
Origin Task	ODTASKID	0000494
Origin Task Start Dat	ODSTARTTM	20160802104739.115607
Origin Task Start Tim	ODSTARTTIME	2016-08-02T10:47:39.115607+00:00
Origin TCPIP Service	ODTCPIPS	CAKE
Origin Transaction II	ODTRANSID	CPIH
Origin User ID	ODUSERID	ALISONB
Origin VTAM LU Nan	ODLUNAME	
◢ Previous hop data		
Prev Hop Appl Id	PHAPPLID	IYK3ZAB3
Prev Hop Count	PHCOUNT	2
Prev Hop Net ID	PHNETWORKID	GBIBMIYA
Prev Hop Start Date	PHSTARTTM	20160802104739.118450
Prev Hop Start Time	PHSTARTTIME	2016-08-02T10:47:39.118450+00:00
Prev Hop Task ID	PHTASKID	0003142
Prev Hop Trans ID	PHTRANSID	CSM3

This shows that the origin of the work in CICS was indeed the task mentioned above, pipeline handler task 494 on region IYK3ZAB4 for URIMAP $649090, and the web service request came in to CICS from IP address 9.20.201.152 port 57145 for user ID ALISONB. If you had noticed the hanging mirror task first, this information could be used to track back to the owner of the work; searching on this mirror's associated tasks would give the same results as searching from the pipeline handler task.

What about work that is not initiated by web service requests? What information is available to identify their points of origin? This depends on the origin itself. For example, work from IBM MQ makes use of adapter data fields of the origin data. The contents of these fields depend on how the work was initiated; for work initiated by the MQ trigger monitor as a result of a message put onto application queue ALITRIG1 on QMGR (queue manager) MQD6, you see the following:



type here to filter on Name and CICS Name

Name	CICS Name	Value
› Basic		
› Current Application Cor		
› Distributed identity infc		
◢ Origin data		
Origin Adapter Data	ODAPTRDATA1	QMGR=MQD6
Origin Adapter Data	ODAPTRDATA2	INITQ=ALIINIT1
Origin Adapter Data	ODAPTRDATA3	QNAME=ALITRIG1
Origin Adapter ID	ODAPTRID	ID=IBM WebSphere MQ for z/OS V7.1
Origin Appl ID	ODAPPLID	IYK3ZAB4
Origin Appl ID Net II	ODNETWORKID	GBIBMIYA

Origin data characteristics

In transaction tracking, every user task in a CICSplex has a point of origin, such as a web service request or an MQ message. When work first enters the CICSplex, details of its point of origin is placed into task context information called *origin data* for the first task that is created to process it. This origin data, flows with the work as it moves around the CICSplex. The origin descriptor record (ODR) is the part of the association data that holds the origin data.

A transaction group ID, TRNGRPID, is the unique key that represents the origin data. You use the TRNGRPID to track where transactions are created when they do not share the unit of work (for example, when you use a **START** command) to indicate which parts of the transaction have a common source. CICS determines the source of information, rather than the target location of the information. Also, with origin data you append your own identifying token to the work request.

Origin data is created when a new request first arrives at a CICS region. This request might be initiated from a web browser, a 3270 terminal, an SNA LU, or another external device. The user task that CICS first attaches is at a new point of origin, and CICS populates the fields in the ODR of this task with information specific to the point of origin. If this task subsequently causes another task to be attached, either in the same region, or in a different region over an IPIC or MRO connection, the origin data is inherited by the new task, unless the new task is at a new point of origin. Figure 4 on page 15 shows how this works. “Example: how origin data is stored and passed for a web request” on page 16 and “Example: how origin data is stored and passed for SNA LU” on page 16 step through the processing of origin data.

A new point of origin is created in the following circumstances:

- A task is attached by a **START** command that specifies the TERMID option.
- A task is attached by a **START** command issued from a PLT program.
- A task is attached by a **START ATTACH** command.
- A task is attached by a DTP or CPIC request.
- A task is attached over an APPC connection.
- A task is attached using the transaction start EP adapter.
- A task is attached in a Liberty JVM server to run a Java™ web application.
- For a Liberty JVM server, when the CICS `ExecutorService.runAsCICS()` method is used from a parent thread that is not running under a CICS task.
- For an OSGi JVM server, when the CICS `ExecutorService.runAsCICS()` method is used, in all cases.
- A web service pipeline handler transaction is routed over an MRO connection.
- An outbound HTTP request is made to CICS using CICS Web support.

If you use CICS Transaction Gateway, the point of origin can be outside CICS (in CICS TG) and the point of origin information is populated to the ODR when the task is started at the boundary of the CICSPlex® SM. For example, CICS TG records context information about the point of origin for the JCA resource adapter, and this information is passed to CICS as part of the origin data.

The origin data fields in the association data all have names that begin with "OD". All fields are populated by CICS, except the user correlator data field, USERCORRDATA, which is a 64-byte area that can be populated by the XAPADMGR global user exit. The exit can be called only from a task that is running at a point of origin in a CICSplex. With origin data, you can track interrelated transactions between regions that use IPIC and MRO connections to share work between them. You can use the CICS Explorer or the CICSPlex SM WUI to search for all the tasks that are active in a CICSplex that share a common set of origin data, or you can search on a subset of the fields.

Origin data is written in monitoring records and stored in CICSPlex SM history records for offline analysis. Origin data is unrecoverable information, which means that the data is not available to any tasks that are attached because of a transaction restart, or with any tasks that are rebuilt from the system log when a region is restarted.

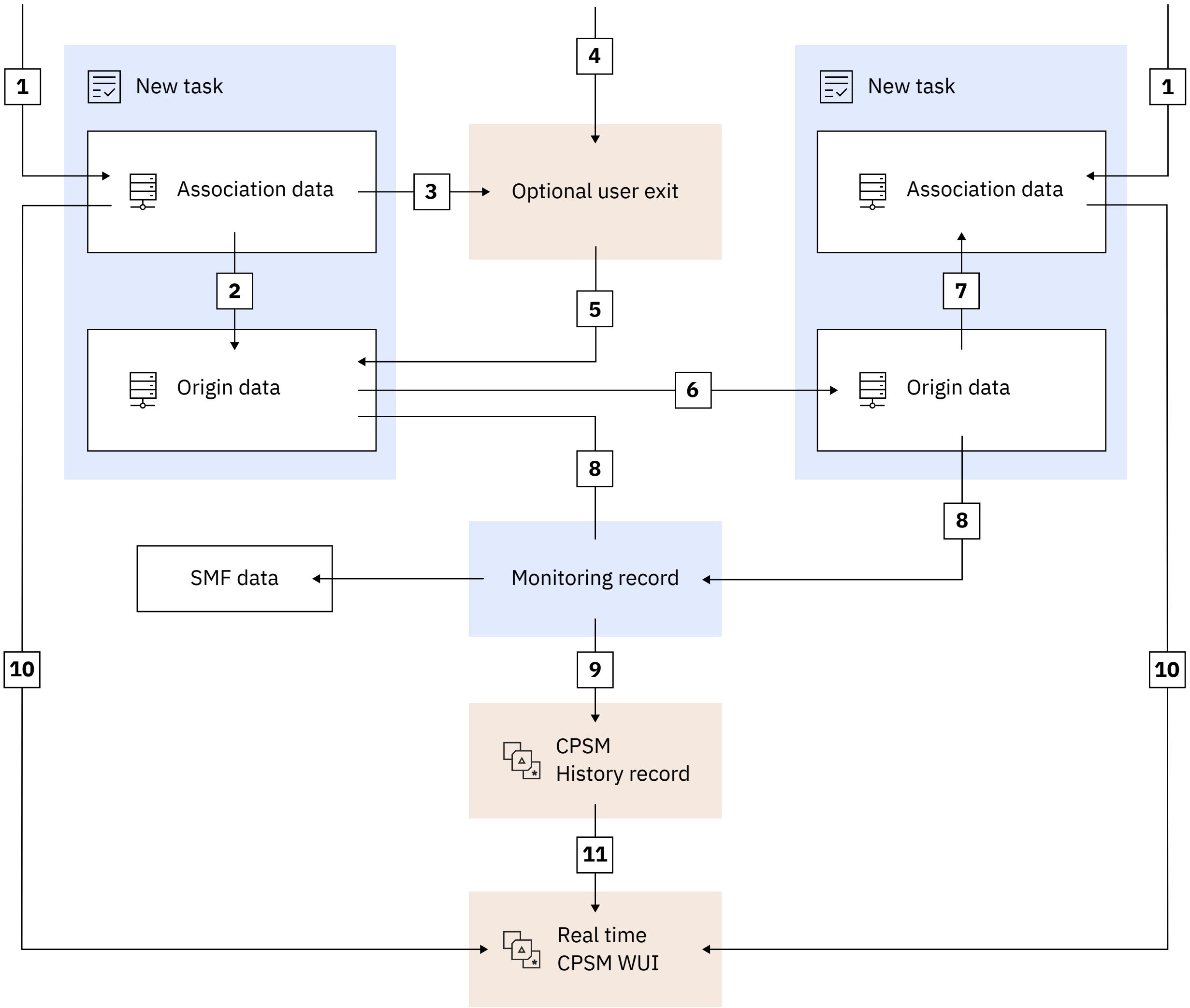


Figure 4. Flow of association data and origin data between CICS tasks and components

- When a new task is attached, association data is created. If the task was created in response to a message received across IP, additional information that CICS obtained from the IP stack 1 is also stored.
- The origin data for the new task is stored in a separate section of the association data 2 and describes where the task was started (the point of origin).
- If a global user exit is called by the task 3, the exit can obtain information from other sources by using the XPI 4 to return to the task 5, where it is included in the origin data.
- If the task issues a DPL request to a remote region, the origin data is added to the DPL request that is sent over TCP/IP to the remote CICS region. When the DPL request arrives at the remote region, another new task is started to process the request. CICS creates unique association data for this task, however CICS detects origin data, and passes the origin data to the mirror task when it is attached to service the DPL request 6.
- During task attach processing, the origin data is stored as part of the association data of the new task, 7, and the global user exit is not called.
- If monitoring is enabled, origin data is written to the monitoring record for the task 8 and if CICSPlex SM is configured, the data is stored in history records 9.
- You can use the CICSPlex SM WUI to retrieve information stored in the association data of running tasks 10; for example, you can create a search to find the tasks in a CICSplex that have matching origin data.
- You can also use CICSPlex SM to perform offline analysis of origin data information that is stored in history records 11; for example, to understand how interrelated transactions have used a Internet Protocol network.

Example: how origin data is stored and passed for SNA LU

A task is started in a region when a transaction identifier is entered at an SNA LU. The origin data is stored at the point of origin and is passed to any other tasks that are started in the same region as a consequence of the initial task:

- The task is at the boundary of the CICSplex and at a point of origin. CICS populates the origin data (SNA LU information) from other fields in its association data when the task is attached.
- If the task issues a DPL request that is serviced in another region using an IPIC connection, the origin data is passed with the DPL request.
- The remote region that receives the message extracts the origin data and passes the data to the mirror transaction, which is attached to service the DPL request.

In this example, the mirror transaction contains the following information in its association data:

- The values that describe the mirror transaction itself; for example, task ID and principal facility of the IPIC connection
- The same origin data that the LU task that scheduled the DPL created and stored in its own association data

In this example, the associated data exit, XAPADMGR, can run when the LU task is attached, but the exit is not called when the mirror task is initialized.

Example: how origin data is stored and passed for a web request

Figure 5 on page 18 shows how origin data is created when CICS processes an HTTP request and how the origin data is inherited by other tasks that are attached to fulfill the request. The HTTP request that has been passed through a TCP/IP network and arrives for CICS processing. The origin data is stored at the point of origin and is passed to any other tasks that are started in the same region as a consequence of the initial task. In this example, the origin data is populated from two different tasks:

1. The HTTP request is passed by a CSOL system task to CICS.
2. The request is processed by a CWXN task. CWXN is at a point of origin and CICS populates the origin data (HTTP request information) from other fields in its association data when the CWXN task

is attached. The XAPADMGR exit can be used to provide origin data in the CWXN task. The exit runs before the HTTP request has been received by CICS so the exit cannot make use of the EXEC CICS **WEB READ HTTPHEADER**, **WEB READ FORMFIELD** or **WEB RECEIVE** commands. This method uses the indirect attach task process. An alternative option is to use the direct attach task process, as the socket listener task CSOL is optimized to directly attach user transactions for fast arriving HTTP requests. This means the web attach task is bypassed, therefore reducing the CPU time that is required to process each request. For more information, see HTTP requests are processed by directly attached user transactions.

3. A new CWBA task is attached and CWBA inherits the ODR from CWXN. CWBA and CWXN might run under different user IDs, but the user ID (userid2) used by the CWBA task is more useful for audit purposes. As a result, the user ID used by the CWBA task is stored in the origin data of CWBA.
4. An application program that is running under the control of the CWBA task issues a DPL request that is serviced over an IPIC connection. The origin data is passed unchanged with the DPL message to the CISR system task.
5. The remote region that receives the DPL message extracts the origin data and passes the origin data to a mirror transaction (CSMI) and the mirror transaction is attached to service the DPL request.
6. The program running under the mirror transaction issues a **START** command. The origin data is inherited by the task (USER) that is attached to service the START request.

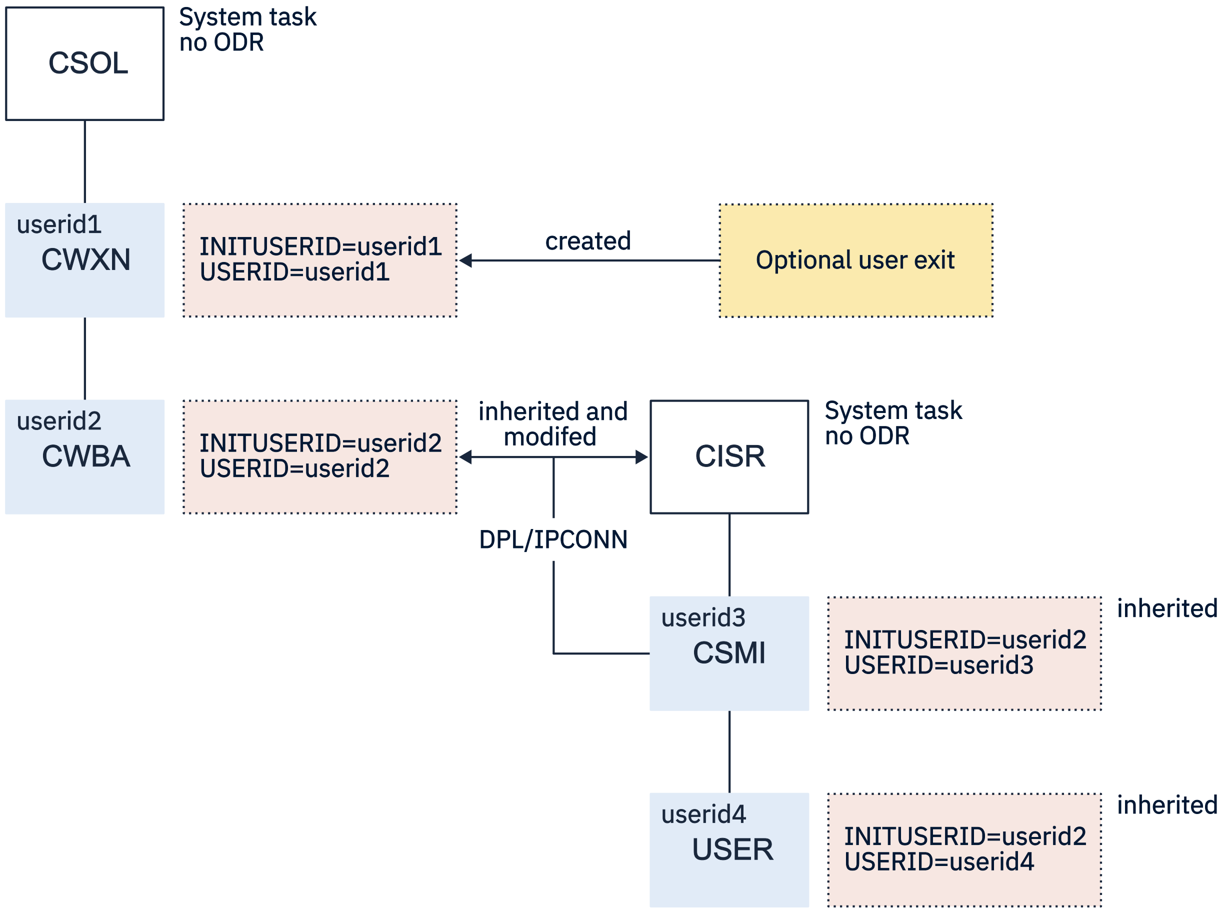


Figure 5. Creation and movement of origin data when an HTTP request is processed

Previous hop data characteristics

Previous hop data identifies the remote sender of a request to attach a task, and creates a trail to be followed back into the previous system, which enables data gathering and monitoring to continue in the region that sent the request.

Previous hop data is created if a request to attach a task is transmitted by using an IPIC or MRO connection between CICS TS 4.2 or later regions. The task that is attached as a result of this request has previous hop data created.

As part of an interrelated transaction, if a task issues requests to attach tasks in other CICS TS 4.2 regions, such as when a daisy chain is used, previous hop data is created for the tasks that are attached in the other CICS regions.

Previous hop data is not created for a task that is the point of origin. For information about association data and the point of origin, see "Association data" on page 8.

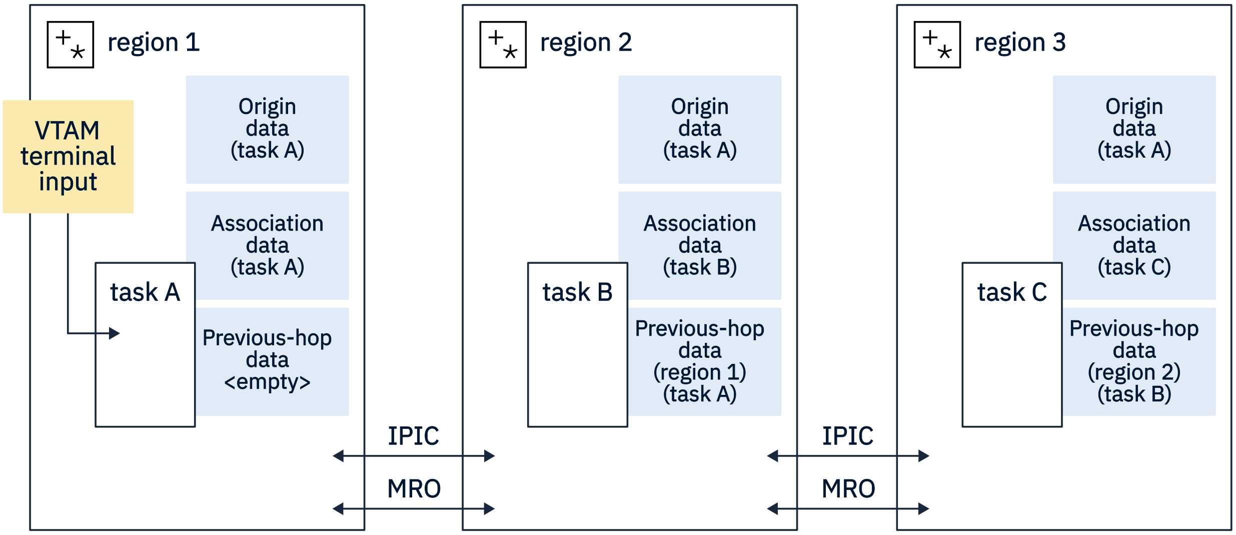

region 1
region 2
region 3
VTAM terminal input
Origin data (task A)
Association data (task A)
task A
Previous-hop data <empty>
Origin data (task A)
Association data (task B)
task B
Previous-hop data (region 1) (task A)
Origin data (task A)
Association data (task C)
task C
Previous-hop data (region 2) (task B)
IPIC
MRO
IPIC
MRO

Figure 6. Previous hop data and an interrelated transaction

The value of previous hop data for a task that is started by use of the **START** command depends on the TERMID option. Previous hop data is not created for a started task that is at a new point of origin.

If the TERMID option is specified, the started task is treated as starting at a point of origin and no previous hop data is created. This is the case whether the TERMID option specifies a local terminal definition or a remote terminal definition.

When the TERMID option specifies a remote terminal definition, the process to schedule the **START** command might involve a transfer of the command across a number of CICS systems to reach the target CICS system where the terminal specified in the TERMID option is a local terminal definition.

If the TERMID option is not specified, only the previous hop count is created for the started task and the remaining previous hop data is not set. In this case, the started task inherits the value of the previous hop count from the task in the same CICS region that initiated it.

For example, if a **START** command without the TERMID option is started by a task that is at a point of origin and the started task runs in the same CICS region, the started task inherits a previous hop count of zero. If the **START** command without the TERMID option is function shipped to another CICS region, the started task inherits a previous hop count from the mirror task.

Previous hop data programming considerations

Previous hop data includes data items that identify the following information:

- Another CICS TS 4.2 or later region that requested the current task to be attached.
- The task in another CICS TS 4.2 or later region that requested the current task to be attached.
- The number of CICS system hops that are taken for all CICS TS 4.2 or later regions to reach the current CICS system. A value of zero is the point-of-origin CICS system.

Previous hop data is not supported when interrelated transactions of a sequence of tasks are run on a number of CICS systems and a previous hop CICS system is a release earlier than CICS TS 4.2. In this case, not all of the previous hop data is set. The previous hop count field is set to one, and no other values in the previous hop data are set.

Previous transaction data characteristics

Previous transaction data identifies the local or parent task of a request to attach a task, for example by a **RUN TRANSID** or **START** command, when a new point of origin is not created. Previous transaction data, when combined with previous hop data, identifies both the local and remote sender of a request to attach a task, and creates a trail that can be followed to the previous task or previous system, which enables data gathering to continue in the region that sent the request.

Previous transaction data helps to manage asynchronously running parent and child tasks — which all run within the same CICS region. Instead of tracking relationships across regions, previous transaction data tracks relationships within a single CICS region, effectively showing you the task depth in any given region, and the parent-child relationships within it. This transaction tracking works for all locally attached tasks, such as those initiated with START commands, as well as tasks initiated by the RUN TRANSID command.

Previous transaction data is created if a request to attach a task is made by a **RUN TRANSID** command, or a **START** command where the **START** command is not a new point of origin. The task that is attached as a result of this request has previous transaction data created. The value of the previous transaction data for a task that is started by the **START** command depends on the TERMID option.

Previous transaction data is not created for a started task that is at a new point of origin. Previous transaction data is not created for a task that is the point of origin.

For a **START** command, the previous transaction data follows the same conventions as for previous hop data. For more information, see “Previous hop data characteristics” on page 19.

Previous transaction data programming considerations

Previous transaction data includes data items that identify the following information:

- The task in the same local region that requested the current task to be attached.
- The current task depth from one task to another in the same CICS system with which this task is associated. A value of zero is the point of origin CICS system, or the first transaction that resulted from a request from one CICS system to another to initiate a task.

ISC and IPIC intercommunications facilities

CICS provides intercommunications facilities for intersystem communication over SNA (ISC over SNA) and IP interconnectivity (IPIC), so that you can communicate with external systems.

This chapter contains the following topics:

- "Intersystem communication over SNA" on page 24
- "Intercommunication using IP interconnectivity" on page 22

Intercommunication using IP interconnectivity

CICS provides intersystem communication over a Transmission Control Protocol/Internet Protocol (TCP/IP) network. This form of communication is called IP interconnectivity or IPIC.

IPIC connection requirements

You must activate TCP/IP services in each CICS region that you are connecting before you create your IPIC connection.

The IPIC connection consists of two complementary resources, an IPCONN definition, and a TCPIPSERVICE definition, which you must install in each CICS region that you are connecting. The IPCONN definition is the CICS resource that represents the outbound TCP/IP communication link, and the term IPCONN is commonly used to refer to an IPIC connection. The inbound attributes of the connection are specified by the TCPIPSERVICE definition. The TCPIPSERVICE resource is named in the TCPIPSERVICE option of the IPCONN definition.

Figure 7 on page 22 shows the relationship between IPCONN and TCPIPSERVICE definitions.

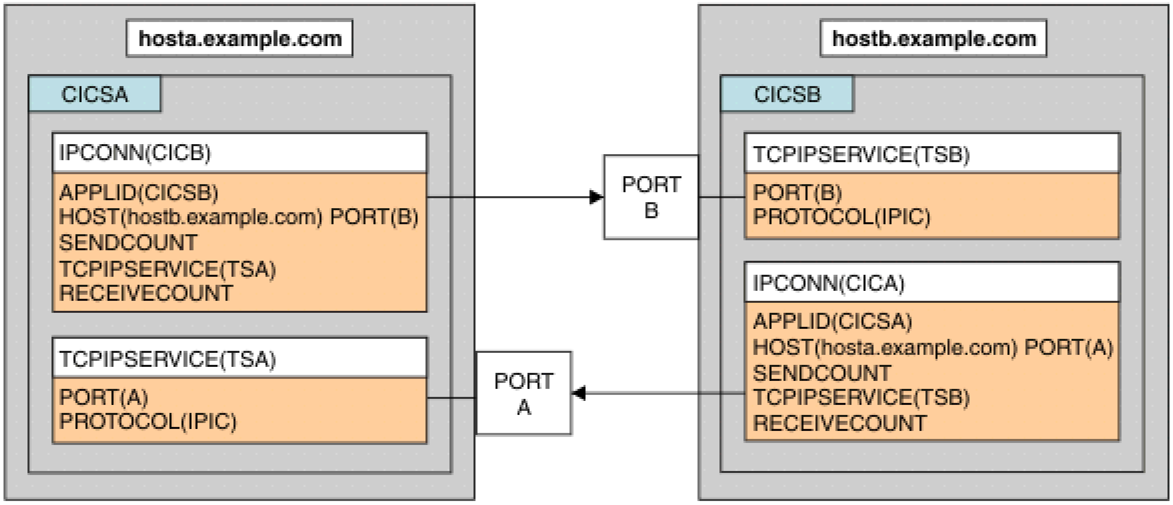


Figure 7. Related IPCONN and TCPIPSERVICE definitions

Synchronization levels

IPIC connections support synchronization level 2; that is, they support full CICS sync pointing, including rollback.

Socket capacity

CICS uses two sockets in each direction (that is, two for outbound and two for inbound) for an IPIC connection. An IPIC connection with CTG uses only the two inbound sockets. If you lose one or more of the sockets that are in use by an IPCONN, for example because of a network error, all the sockets are lost and the IPCONN is released.

CICS IPIC heartbeat function

The CICS IPIC heartbeat function discovers and reports network connectivity problems with idle connections by sending a heartbeat message over a connection during periods when no other messages are being transmitted. Before the introduction of the CICS IPIC heartbeat, connectivity problems with an idle connection remained unnoticed until an attempt was made to send a new message over the connection.

The CICS IPIC heartbeat also maintains a connection that might pass through one or more firewalls. If the CICS IPIC heartbeat is not used, the connection can time out because of lack of usage.

The CISP task checks IPCONN resources at intervals of approximately 60 seconds. An IPCONN resource is selected for the CICS IPIC heartbeat function if the connected CICS region is in a different sysplex. A CIS1 task is attached to resources that are acquired, and are then connected to a CICS region that can accept a heartbeat message, and have received no message traffic since the end of the previous interval.

If the heartbeat message is sent and a response is not received within 10 seconds, a second message is sent. If a response to the heartbeat message is not received within a further 10 seconds, the connection is put into a released state. If a heartbeat message cannot be sent because the connection is no longer usable, an error message is issued and the connection is released automatically.

CICS IPIC heartbeat messages are only sent by CICS TS V5.1 regions or higher; however, they can be received by older releases of CICS TS at V4.1 or higher. The heartbeat messages cannot be sent to any version of CICS Transaction Gateway or TXSeries.

IPIC high availability feature

The IPIC high availability capability provides a single point of access to a cluster of CICS TS regions via a TCP/IP network

Overview

The IPIC high availability capability provides a single point of access to a cluster of CICS TS regions via a TCP/IP network. This ensures resilience of access to the cluster as a whole, for both planned and unplanned outages of individual regions within the cluster. It also supports the ability for client regions that have lost contact with a specific region within a cluster to reconnect back to the specific HA cluster region so that any unit of work affinities can be resolved. The single end point of the cluster is managed on z/OS by a connection balancing mechanism that spreads connectivity across the set of regions within a cluster. z/OS Communications Server offers two methods to achieve this: TCP/IP port sharing and Sysplex Distributor. However IPIC HA is not limited to the use of either of these. See Connection balancing for more information about port sharing and Sysplex Distributor.

How it works

HA cluster regions listen on two end points, defined via TCPIPSERVICE resources. The generic end point that is shared by all regions in the cluster and a specific end point that is used exclusively by a specific region. The cluster regions can have a set of IPCONN resources for the client regions that may attempt to connect to the generic end point, or can make use of the IPCONN autoinstall mechanism.

A client region connects to the HA cluster using the generic end point information defined in an HA IPCONN resource. The connection request to the generic end point will be intercepted by the connection balancing mechanism being used, and routed to a generic TCPIPSERVICE belonging to one of the HA cluster regions. The cluster region then returns the IP address and port of its specific end point to the client region, and the connection is established using the specific end point. See Defining IP interconnectivity (IPIC) high availability (HA) connections.

IPIC HA is only available using CICS TS for z/OS, Version 5.2 or later regions, both within the cluster and those connecting to it. If a back level region connects to an HA cluster region's generic TCPIPSERVICE, the request will be rejected. If a back level region connects to the specific TCPIPSERVICE of an HA cluster region the request will be treated as a non-HA IPIC connection.

Recovery in an IPIC HA environment

A connection into a cluster may fail, leaving units of work (UOWs) that need to be resynchronised at a later point.

When a client region, with outstanding units of work, tries to reconnect to an HA cluster then an attempt is made to restore the connection to the specific region in the cluster that these UOWs relate to. If successful then resynchronisation processing takes place to complete these UOWs. A client region can suffer multiple connection failures, leaving more than one set of UOWs that require resynchronising. If the client region is unable to connect to a cluster region associated with one set of outstanding UOWs, then it will look to see if there is another set of UOWs that need processing and attempt to connect to their associated cluster region. This sequence of events continues until a connection is established with a specific server region, or until there are no more sets of UOWs found, at which point the client region will then attempt to connect to the HA cluster's generic end point.

The client region's IPCONN XLNACTION attribute will be used to decide how any outstanding UOWs are dealt with once a connection becomes acquired. This attribute defaults to KEEP, meaning that any UOWs that could not be resynchronised are retained. If XLNACTION is set to FORCE then those UOWs that could not be resynchronised at that time are forced to complete heuristically, according to the decision indicated by the ACTION attribute of their associated TRANSACTION resource definition.

A client region that has no outstanding UOWs will always connect to the HA cluster's generic end point.

Intercommunication facilities available using IPIC

IP interconnectivity (IPIC) supports communication between CICS systems using a TCP/IP network.

IPIC supports the following types of intercommunication functions for their respective product releases:

- Distributed program link (DPL) calls between CICS regions and between CICS TS and TXSeries.
- Asynchronous processing of **EXEC CICS START**, **START CHANNEL**, and **CANCEL** commands between CICS regions.
- Transaction routing of 3270 terminals, where the terminal-owning region (TOR) is uniquely identified by an APPLID between CICS regions.
- Enhanced method of routing transactions that are invoked by **EXEC CICS START** commands between CICS regions.
- ECI requests from CICS Transaction Gateway.
- Function shipping of all file control, transient data, and temporary storage requests between CICS regions. Function shipping using IPIC connectivity is threadsafe between CICS regions at a supported release level.
- Threadsafe processing for the mirror program and the LINK command to improve performance for threadsafe applications.
- RESTful API calls through z/OS Connect Enterprise Edition

Intersystem communication over SNA

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

Before reading these topics, you must be familiar with the general concepts and terminology of SNA.

This chapter contains the following topics:

- “Connections between subsystems” on page 25
- “Intersystem sessions” on page 27
- “Establishing intersystem sessions” on page 29.

Intercommunication facilities available using ISC

Intersystem communication over SNA (ISC over SNA) allows communication between CICS and non-CICS systems or CICS systems that are not in the same z/OS image or sysplex. These intercommunication facilities can also be used between CICS regions in the same z/OS image or sysplex.

These facilities are available for intercommunication using ISC:

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

ISC can be used between CICS and any other system that supports the z/OS Communications Server Advanced Program-to-Program Communication (APPC) or SNA Logical Unit Type 6.1 (LUTYPE6.1) communications. For example, ISC over SNA connections can exist between CICS regions running in different z/OS sysplexes or on different operating system platforms, between CICS and any APPC device, and between CICS and IMS.

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

- Other CICS Transaction Server for z/OS systems
- CICS Transaction Server for VSE
- CICS Transaction Server for iSeries
- IMS Version 9.1 or later
- Any system that supports Advanced Program-to-Program Communication (APPC) protocols (LU6.2)

Connections between subsystems

Subsystems can be connected for intersystem communication in three basic forms.

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

A possible configuration is shown in Figure 8 on page 26.

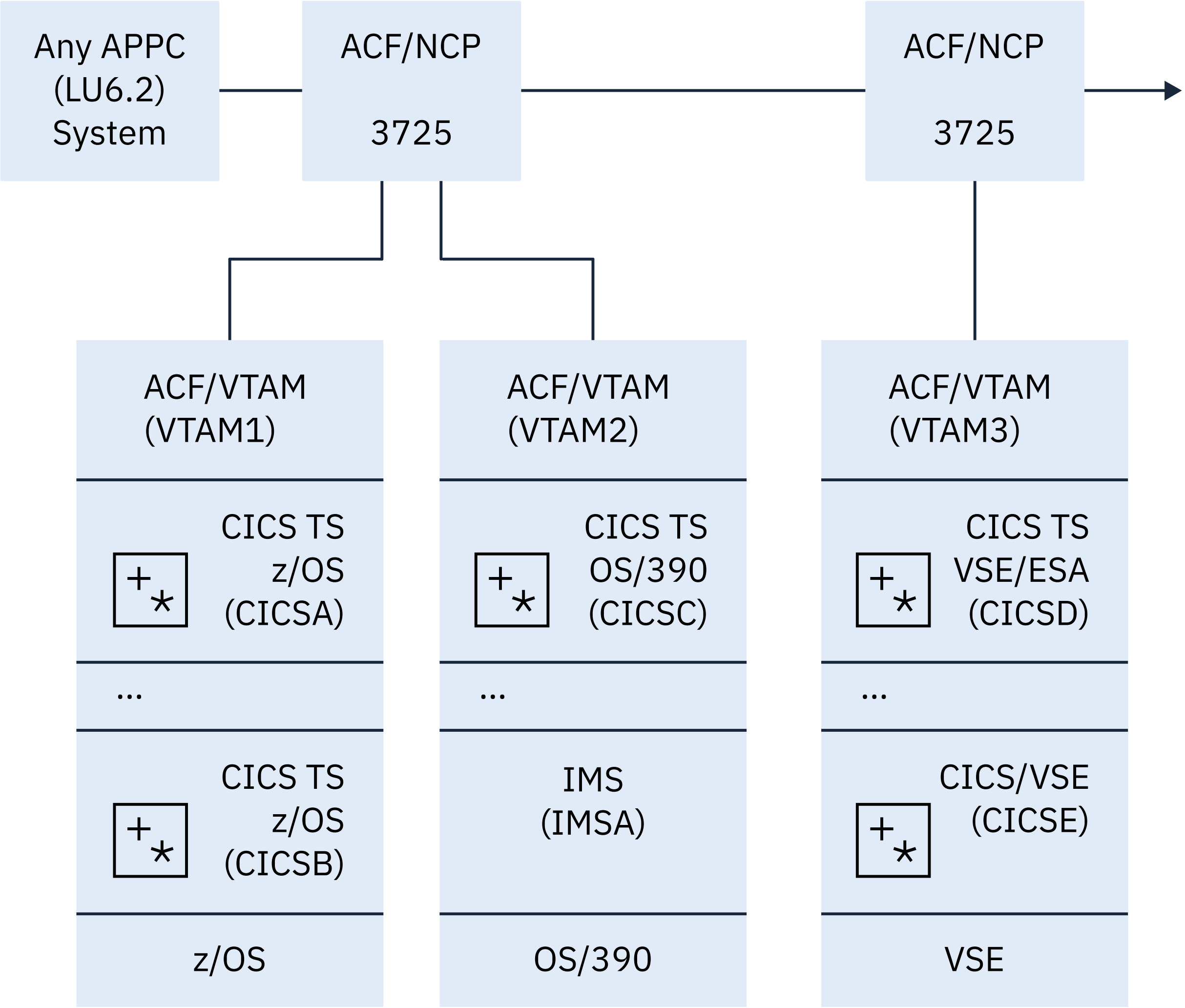


Figure 8. A possible configuration for intercommunicating systems

Single operating system

ISC in a single operating system (intrahost ISC) is possible through the application-to-application facilities of ACF/SNA. In Figure 8 on page 26, these facilities can be used to communicate between CICSA and CICSB, between CICSC and IMSA, and between CICSD and CICSE.

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

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

Physically adjacent operating systems

You can configure an IBM 3725 with a multichannel adapter that permits you to connect two SNA domains (for example, VTAM1 and VTAM2 in Figure 8 on page 26) through a single ACF/NCP/VS. This configuration might be useful for communication between these systems:

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

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

Remote operating systems

The most typical configuration for intersystem communication is between remote operating systems. For example, in Figure 8 on page 26, CICSD and CICSE can be connected to CICSA, CICSB, and CICSC in this way. Each participating system is appropriately configured for its particular location, using MVS or Virtual Storage Extended (VSE) CICS or IMS, and one of the ACF access methods such as ACF/SNA.

For a list of the CICS and non-CICS systems to which CICS Transaction Server for z/OS can connect to using ISC, see “Communication between systems” on page 1.

Intersystem sessions

CICS uses ACF/SNA to establish, or *bind*, logical-unit-to-logical-unit (LU-LU) sessions with remote systems. Being a logical connection, an LU-LU session is independent of the physical route between the two systems. A single logical connection can carry multiple independent sessions. Such sessions are called *parallel* sessions.

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

- LUTYPE6.1 sessions
- LUTYPE6.2 generally called APPC sessions.

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

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

LUTYPE6.1

LUTYPE6.1 is the forerunner of LUTYPE6.2 (APPC).

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

LUTYPE6.2 (APPC)

The general term used for the LUTYPE6.2 protocol is Advanced Program-to-Program Communication (APPC). In addition to enabling data communication between transaction-processing systems, the APPC architecture defines subsets that enable device-level products (APPC terminals) to communicate with host-level products and also with each other.

You can use APPC sessions for CICS-to-CICS communication and for communication between CICS and other APPC systems or terminals.

Here is an overview of some of the principal characteristics of the APPC architecture.

Protocol boundary

The APPC protocol boundary is a generic interface between transactions and the SNA network. It is defined by formatted functions, called *verbs*, and protocols for using the verbs. Details of this SNA protocol boundary are in z/OS Communications Server: SNA Programmer's LU 6.2 Guide.

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

Two types of APPC conversation are defined:

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

Basic
In basic conversations, the data passed to and received from the APPC application program interface is prefixed with a header, called a GDS header. The user is responsible for building and interpreting this header. Basic conversations are used principally for communication with device-level products that do not support mapped conversations, and which possibly do not have an application programming interface open to the user.

Synchronization levels

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

Level 0 (NONE)
This level is for use when communicating with systems or devices that do not support synchronization points, or when no synchronization is required.

Level 1 (CONFIRM)
This level allows conversing transactions to exchange private synchronization requests. CICS built-in synchronization does not occur at this level.

Level 2 (SYNCPOINT)
This level is the equivalent of full CICS syncpointing, including rollback. Level 1 synchronization requests can also be used.

EXEC CICS commands and CPI Communications support all three levels.

Program initialization parameter data

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

LU services manager

Multisession APPC connections use the LU services manager, the software component responsible for negotiating session binds, session activation and deactivation, resynchronization, and error handling. It requires two special sessions with the remote LU; these are called the *SNASVCMG sessions*. When these sessions are bound, the two sides of the LU-LU connection can communicate with each other, even if the connection is 'not available for allocation' for users.

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

Class of service

The CICS implementation of APPC includes support for “class of service” selection.

Class of service (COS) is an ACF/SNA facility that allows sessions between a pair of logical units to have different characteristics.

- Alternate routing: virtual routes for a given COS can be assigned to different physical paths (explicit routes).
- Mixed traffic: different kinds of traffic can be assigned to the same virtual route and, by selecting appropriate transmission priorities, undue session interference can be prevented.
- Trunking: explicit routes can use parallel links between specific nodes.

In particular, sessions can take different virtual routes, and thus use different physical links; or, the sessions can be of high or low priority to suit the traffic carried on them.

In CICS, APPC sessions are specified in groups called *modesets*, each of which is assigned a *modename*. The modename must be the name of a z/OS Communications Server SNA LOGMODE entry (also called a *modegroup*), which can specify the class of service required for the session group. For more information, see ACF/Communications Server LOGMODE table entries for CICS.

Limited resources

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

Both single- and multi-session connections can use limited resources. For a multi-session connection, CICS does not unbind LU service-manager sessions until all modegroups in the connection have performed initial "change number of sessions" (CNOS) exchange. When CICS unbinds a session, CICS tries to balance the contention winners and losers. This balancing might result in CICS resetting an unbound session to be neither a winner or a loser.

Establishing intersystem sessions

Before traffic can flow on an intersystem session, the session must be established, or *bound*.

CICS can be either the primary (BIND sender) or secondary (BIND receiver) in an intersystem session, and can be either the contention winner or the contention loser. The contention winner in an LU-LU session is the LU that is permitted to begin a conversation at any time. The contention loser is the LU that must use an SNA BID command (LUTYPE6.1) or LUSTATUS command (APPC) to request permission to begin a conversation.

You can specify the number of contention-winning and contention-losing sessions required on a link to a particular remote system.

For LUTYPE6.1 sessions, CICS always binds as a contention loser.

For APPC links, the number of contention-winning sessions is specified when the link is defined. See Defining APPC links. The contention-winning sessions are normally bound by CICS, but CICS also accepts bind requests from the remote system for these sessions.

Normally, the contention-losing sessions are bound by the remote system. However, CICS can also bind contention-losing sessions if the remote system is cannot send bind requests.

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

Session initiation occurs in one of the following ways:

- By CICS during CICS initialization for sessions for which AUTOCONNECT(YES) or AUTOCONNECT(ALL) has been specified. See Defining connections to remote systems.
- By a request from the CICS main terminal operator.
- By the remote system with which CICS is to communicate.
- By CICS when an application explicitly or implicitly requests the use of an intersystem session and the request can be satisfied only by binding a previously unbound session.

Multiregion operation

By using CICS multiregion operation (MRO), CICS systems that are running in the same MVS image, or in the same MVS sysplex, can communicate with each other.

This chapter contains the following topics:

- "Intercommunication facilities available using MRO" on page 30
- "Cross-system multiregion operation (XCF/MRO)" on page 31
- "Applications of multiregion operation" on page 36
- "Conversion from a single-region system" on page 38.

Intercommunication facilities available using MRO

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

MRO provides these intercommunication facilities:

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

MRO has some restrictions for distributed transaction processing. The external CICS interface (EXCI) uses a special form of MRO link to support these types of communication between MVS batch programs and CICS.

MRO does not need networking facilities. CICS support for region-to-region communication is called *interregion communication* (*IRC*). You can implement IRC in three ways:

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

CICS regions linked by MRO can be at different release levels. If an MVS image contains different releases of CICS, all using MRO to communicate with each other or XCF/MRO to communicate with regions in other images in the sysplex, the DFHIRP module in the MVS LPA must be from the most current CICS release in the image, or higher.

Cross-system multiregion operation (XCF/MRO)

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

IRC provides an XCF access method that makes it unnecessary to use z/OS Communications Server to communicate between MVS images within the same MVS sysplex.

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

When members of a CICS XCF group that are in different MVS images communicate, CICS selects the XCF access method dynamically, overriding the access method specified on the connection resource definition. By means of MVS cross-system coupling facility, MRO can function *between* MVS images in a sysplex environment, supporting all the usual MRO operations.

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

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

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

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

CICS regions linked by XCF/MRO can be at different release levels; see “Multiregion operation” on page 1. Depending on the versions of CICS installed in the MVS images participating in XCF/MRO, the versions of DFHIRP installed in the link pack areas of the MVS images can be different. If a single MVS image contains different releases of CICS, all using XCF/MRO to communicate with regions in other images in the sysplex, the DFHIRP module in the MVS LPA must be that from the most current CICS release in the image, or higher. However, note that the CICS TS for z/OS, Version 4.1 version of DFHIRP (required for multiple XCF group support) can be used only on z/OS Version 1.7 or later. For full details of software and hardware requirements for XCF/MRO, see Installation requirements for XCF/MRO.

Figure 9 on page 33 is an example of the use of XCF/MRO in a sysplex environment. This example, has only one CICS XCF group, DFHIR000. The members of DFHIR000 can communicate using XCF/MRO links across the two MVS images.

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

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

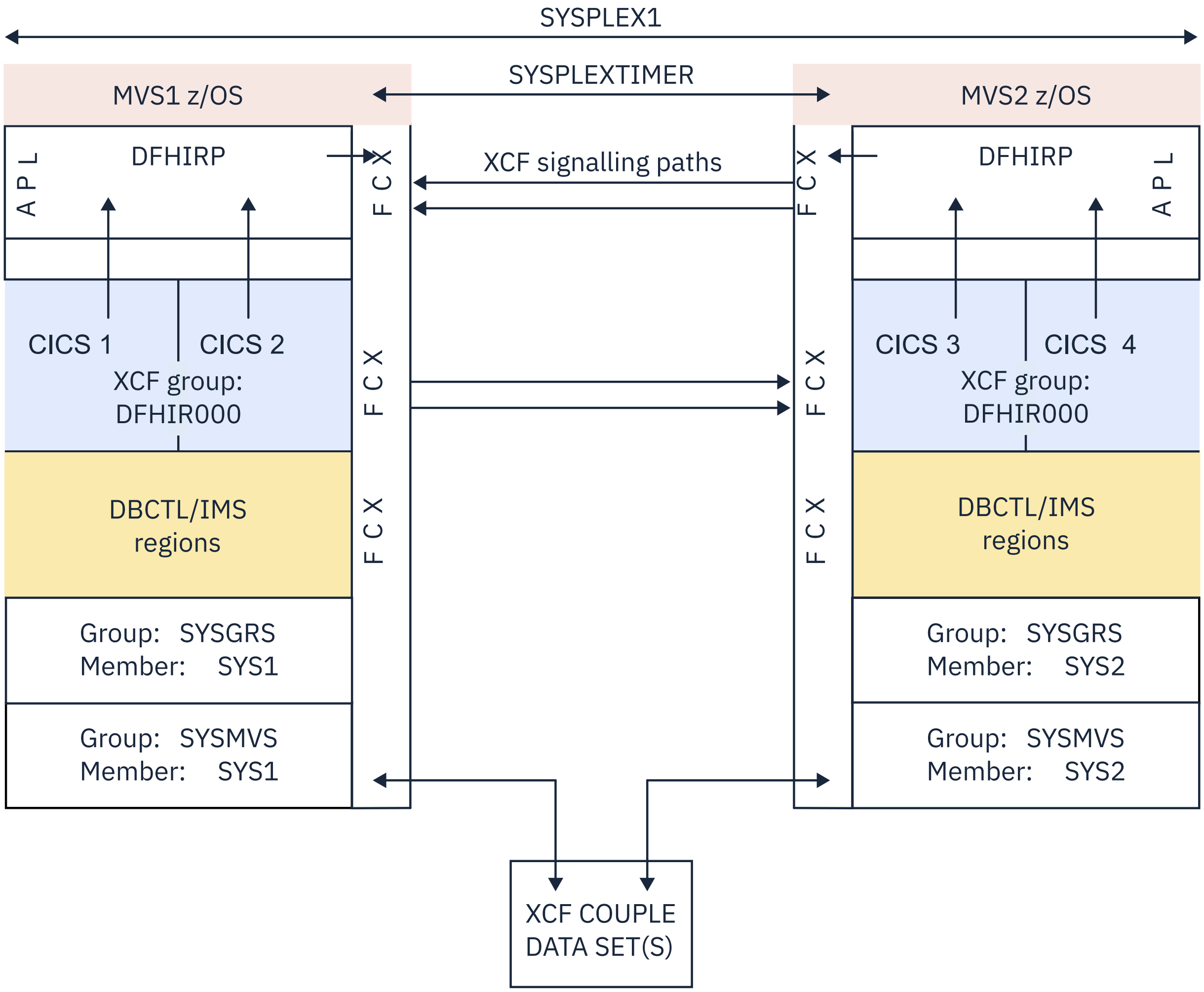


Figure 9. A sysplex (SYSPLEX1) containing a single CICS XCF group

Figure 10 on page 35 is a slightly more complex example. This example has two CICS XCF groups, DFHIR000 and DFHIR001. The members of each XCF group can communicate across the MVS images by means of XCF/MRO links.

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

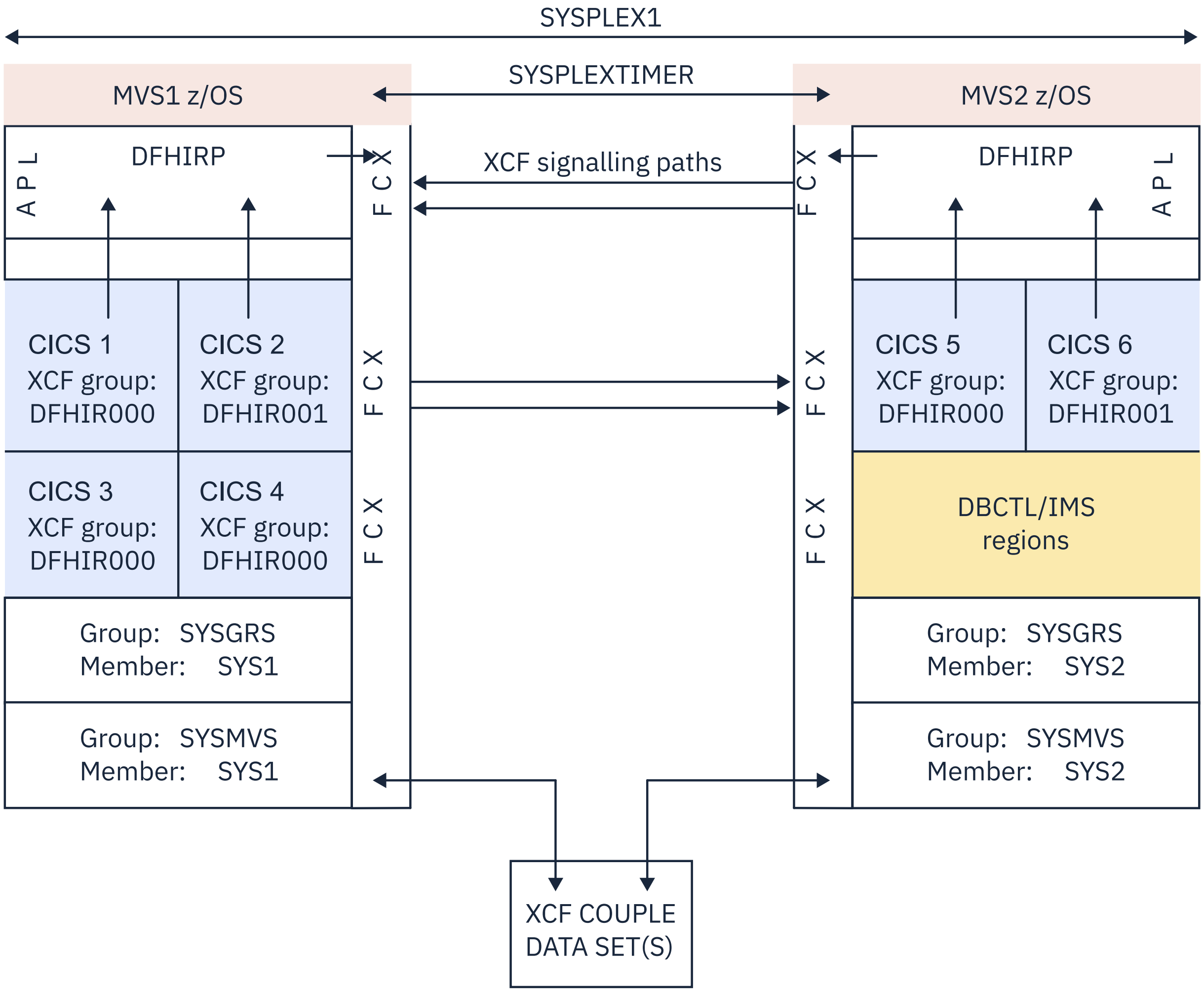


Figure 10. A sysplex (SYSPLEX1) containing two CICS XCF groups

Note:

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

Benefits of XCF/MRO

Cross-system MRO using XCF links offers a number of benefits.

- A low communication overhead between MVS images, providing much better performance than using ISC links to communicate between MVS systems. XCF/MRO thus improves the efficiency of transaction routing, function shipping, asynchronous processing, and distributed program link across a sysplex. You can also use XCF/MRO for distributed transaction processing if the LUTYPE6.1 protocol is adequate for your purpose.
- Easier connection resource definition than for ISC links, with no SNA (z/OS Communications Server) tables to update.
- Good availability, by having alternative processors and systems ready to continue the workload of a failed MVS or a failed CICS.
- Easier transfer of CICS systems between MVS images. The more straightforward connection resource definition of MRO, with no SNA tables to update, makes it easier to move CICS regions from one MVS to another. You no longer need to change the connection definitions from CICS MRO to CICS ISC (which can be done only if the CICS startup on the new MVS is a warm or cold start).
- Improved price and performance, by coupling low-cost, rack-mounted, air-cooled processors in an HPCS environment.
- Growth in small increments.
- Organizational benefits. Because regions in different XCF groups cannot communicate over MRO or XCF/MRO, each group of regions is effectively isolated from the others. This isolation can be useful if, for example, you want to prevent, possibly, access accidentally to production regions from development or test regions.

Applications of multiregion operation

Multiregion operation provides an environment for a number of typical applications.

Program development

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

You can start and stop the test system as required, without interrupting production work. During the cutover of the new programs into production, terminal operators can run transactions in the test system from their regular production terminals, and the new programs can access the full resources of the production system.

Time-sharing

If one CICS system performs compute-bound work, such as APL or ICCF, as well as regular DB/DC work, the response time for the DB/DC user can be unduly long. You can improve the response time by running the compute-bound applications in a lower-priority address space and the DB/DC applications in another address space.

Transaction routing allows any terminal to access either CICS system without the operator being aware of the two different systems.

Reliable database access

You can use CICS storage protection and transaction isolation to guard against unreliable applications that might otherwise stop the system or disable other applications.

However, you might use MRO to extend the level of protection.

For example, you might define two CICS regions, one that owns applications that you have identified as unreliable, and the other that owns the reliable applications and the database. If you run a smaller number of applications in the database-owning region, you have a more reliable region. However, the cross-region traffic is greater, so performance can be degraded. You must balance performance against reliability.

You can take this application of MRO to its limit by having no user applications at all in the database-owning region. The online performance degradation might be a worthwhile trade-off against the elapsed time necessary to restart a CICS region that owns a very large database.

Departmental separation

MRO enables you to create a *CICSplex* in which the various departments of an organization have their own CICS systems.

Each can start and end its own system as it requires. At the same time, each can have access to other departments' data, with access controlled by the system programmer. A department can run a transaction on another department's system, again subject to the control of the system programmer. Terminals need not be allocated to departments, because, with transaction routing, any terminal can run a transaction on any system.

Multiprocessor performance

Using MRO, you can take advantage of a multiprocessor by linking several CICS systems into a CICSplex, and allowing any terminal to access the transactions and data resources of any of the systems.

The system programmer can assign transactions and data resources to any of the connected systems to get optimum performance. Transaction routing presents the terminal user with a single system image; the user is not aware that more than one CICS system is present.

Transaction routing is described in "CICS transaction routing" on page 67.

Workload balancing in a sysplex

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

You can use these products and functions to perform workload balancing:

- The z/OS Communications Server generic resource function
- Dynamic transaction routing
- Dynamic routing of DPL requests
- CICSPlex System Manager (CICSPlex SM)
- The z/OS Workload Manager (WLM)

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

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

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

- “Dynamically routing DPL requests” on page 103
- “Dynamic transaction routing” on page 70

Virtual storage constraint relief

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

In such cases, you might be able to relieve the virtual storage problem by splitting the system into two or more separate systems with shared resources. You can use all the facilities of MRO to help maintain a single-system image for users.

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

Conversion from a single-region system

Usually, you can convert existing single-region CICS systems to multiregion CICS systems with little or no reprogramming.

CICS function shipping allows operators of terminals owned by an existing command-level application to continue accessing existing data resources after either the application or the resource has been transferred to another CICS region. Applications that use function shipping must follow the rules given in Application programming for CICS function shipping. To conform to these rules, you might have to modify programs written for single-region CICS systems.

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

You must define an MRO link between the two regions and to provide local and remote definitions of the shared resources.

CICS function shipping

You can use CICS function shipping to write CICS application programs without regard to the location of the requested resources. They use file control commands, temporary-storage commands, and other functions in the same way.

This chapter contains the following topics:

- “Overview of function shipping” on page 39
- “Design considerations for Function Shipping” on page 41
- “The mirror transaction and transformer program” on page 43

- “Function shipping examples” on page 47.

Overview of function shipping

You can use CICS function shipping to enable CICS application programs to perform the following tasks.

- Access CICS files owned by other CICS systems by shipping file control requests.
- Access DL/I databases managed by or accessible to other CICS systems by shipping requests for DL/I functions.
- Transfer data to or from transient data and temporary storage queues in other CICS systems by shipping requests for transient data and temporary storage functions.
- Initiate transactions in other CICS systems, or other non-CICS systems that implement SNA LU Type 6 protocols, such as IMS, by shipping interval control START requests. This form of communication is described in “Asynchronous processing” on page 50.

You can write applications without regard to the location of the requested resources. They use file control commands, temporary-storage commands, and other functions in the same way. Entries in the CICS resource definition tables allow the system programmer to specify that the named resource is not on the local (or requesting) system but on a remote (or owning) system.

An illustration of a shipped file control request is given in Figure 11 on page 40. In this figure, a transaction running in CICA issues a file control READ command against a file called NAMES. The resource definition for the file indicates that this file is owned by a remote CICS system called CICB. CICS changes the READ request into a suitable transmission format and then ships it to CICB for execution.

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

CICS recovery and restart enables resources in remote systems to be updated, and ensures that, when the requesting application program reaches a synchronization point, any mirror transactions that are updating protected resources also take a synchronization point, so that changes to protected resources in remote and local systems are consistent. The CICS main terminal operator is notified of any failures in this process, so that suitable corrective action can be taken. This action can be taken manually or by user-written code.

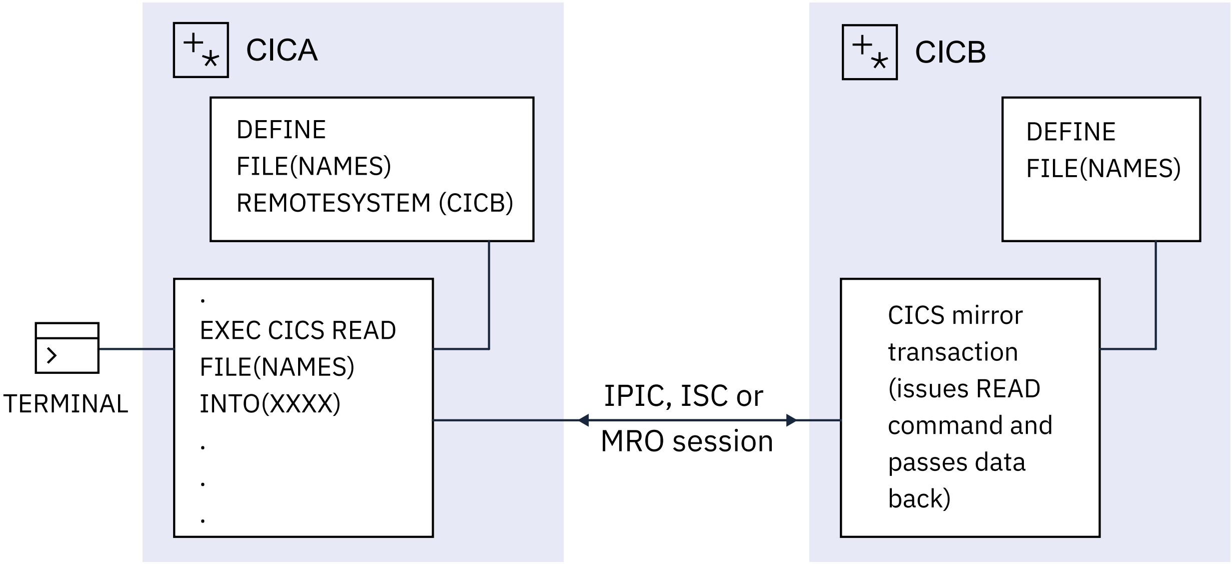


Figure 11. Function shipping

Design considerations for Function Shipping

User application programs can run in a CICS intercommunication environment and use the intercommunication facilities without being aware of the location of the file or other resource that is being accessed. The location of the resource is specified in the resource definition.

Guidance on identifying and defining remote resources is given in Defining remote resources.

The resource definition can also specify the name of the resource as it is known on the remote system, if it is different from the name by which it is known locally. When the resource is requested by its local name, CICS substitutes the remote name before it sends the request. Substituting the remote name is useful when a particular resource exists with the same name on more than one system but contains data specific to the system on which it is located.

This technique might limit program independence. Application programs can also name remote systems explicitly on commands that can be function-shipped, by using the SYSID option. If you specify this option, the request is routed directly to the named system, and the resource definition tables on the local system are not used. You can specify the local system in the SYSID option so that the decision whether to access a local resource or a remote one can be taken at execution time.

For design considerations that relate to initiating transactions in other CICS systems, or other systems that are not CICS but implement SNA LU Type 6 protocols, by shipping interval control **START** requests, see "Asynchronous processing" on page 50.

File control

Function shipping allows access to VSAM or BDAM files located on a remote CICS system.

Note the following points:-

- INQUIRE FILE, INQUIRE DSNAME, SET FILE, and SET DSNAME are not supported.
- Both read-only and update requests are allowed, and the files can be defined as protected on their own system.
- Updates to remote protected files are not committed until the application program issues a sync point request or terminates successfully.
- Linked updates of local and remote files can be performed in the same unit of work, even if the remote files are located on more than one connected CICS system.

Important:

Take care when designing systems in which remote file requests using physical record identifier values are employed, such as VSAM RBA, BDAM, or files with keys not embedded in the record. You must ensure that all application programs in remote systems have access to the correct values following addition of records or reorganization of these types of file.

DL/I

Function shipping allows a CICS transaction to access IMS Database Manager (IMS DM) databases associated with a remote CICS system, or DL/I databases associated with a remote CICS Transaction Server for VSE system.

See Chapter 1, "CICS intercommunication," on page 1 for a list of systems with which CICS Transaction Server for z/OS, Version 5 Release 6 can communicate.

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

As with file control, updates to remote DL/I databases are not committed until the application reaches a sync point. With IMS DM, it is not possible to schedule more than one program specification block (PSB) for each unit of work, even when the PSBs are defined to be on different remote systems. Therefore linked DL/I updates on different systems cannot be made in a single unit of work.

The PSB directory list (PDIR) is used to define a PSB as being on a remote system. The remote system owns the database and the associated program communication block (PCB) definitions.

Temporary storage

Function shipping enables application programs to send data and retrieve data from temporary storage queues located on remote systems.

You can define a remote temporary storage queue by specifying remote attributes in a TSMODEL resource definition. If the queue is to be protected, you must define it as recoverable.

Transient data

An application program can access intrapartition or extrapartition transient-data queues on remote systems.

The definition of the queue in the requesting system defines it as being on the remote system. The definition of the queue in the remote system specifies its recoverability attributes, and whether it has a trigger level and associated terminal. You can define extrapartition queues in the owning system as having records of fixed or variable length.

Many current uses of transient-data and temporary-storage queues can be extended to an interconnected processor system environment. For example, you can create a queue of records in a system for processing overnight. Queues also provide another means of handling requests from other systems while freeing the terminal for other requests. The reply can be returned to the terminal when it is ready, and delivered to the operator when there is a lull in entering transactions.

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

By means of the remote naming capability, a program can send data to the CICS service destinations, such as CSMT, in both local and remote systems.

Intersystem queuing

Performance problems can occur when function shipping requests that are waiting for free sessions are queued in the issuing region.

Requests that are to be function shipped to a resource-owning region might be queued if all bound contention winner sessions are busy, so that no sessions are immediately available. If the resource-owning region is unresponsive, the queue can become so long that the performance of the issuing region is severely impaired. Further, if the issuing region is an application-owning region, its impaired performance can spread back to the terminal-owning region.

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

On IPIC connections, queuing starts when there are no available send sessions. The number of send sessions are specified using the SENDCOUNT attribute on the IPCONN resource definition on the local server. The number of receive sessions are specified using the RECEIVECOUNT attribute defined in the IPCONN resource definition on the remote system. The number of send sessions that are used is the lower of the following two values:

- SENDCOUNT on the local definition
- RECEIVECOUNT on the remote definition

The symptoms of this impaired performance are as follows:

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

In either case, CICS cannot start any new work.

CICS provides two methods to prevent these problems:

- The QUEUELIMIT and MAXQTIME options on both the IPCONN and CONNECTION definitions. You can use these options to limit the number of requests that can be queued against particular remote regions, and the time that requests must wait for sessions on unresponsive connections.
- The global user exits XZIQUE, XISCONA, and XISQUE. The XZIQUE or XISCONA exit program is invoked if no contention winner session is immediately available. The exit program can instruct CICS to queue the request, or to return SYSIDERR to the application program. Its decision can be based on statistics accessible from the user exit parameter list. For programming information about writing XZIQUE and XISCONA exit programs, refer to Intersystem communication program exits, XISCONA, XISLCLQ, and XISQLCL. For information about the statistics records that are passed to your exit program, refer to Introduction to CICS statistics. The global user exit XISQUE is used to manage IPIC intersystem queues. See XISQUE exit for managing IPIC intersystem queues.

  Note: For non-IPIC connections, it is best practice to use the XZIQUE exit, rather than XISCONA. XZIQUE provides better function, and is of more general use than XISCONA. It is driven for function shipping, DPL, transaction routing, and distributed transaction processing requests, whereas XISCONA is driven only for function shipping and DPL. If you enable both exits, XZIQUE and XISCONA can both be driven for function shipping and DPL requests, which is not recommended.

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

For further information about controlling intersystem queues, see Intersystem session queue management.

The mirror transaction and transformer program

CICS supplies a number of mirror transactions, some of which correspond to "architected processes".

Details of the supplied mirror transactions are given in Defining local resources. Here, they are referred to generally as the mirror transaction and have the transaction identifier CSM*.

The mirror transaction runs as a normal CICS transaction and is threadsafe when an IPIC connection is used.

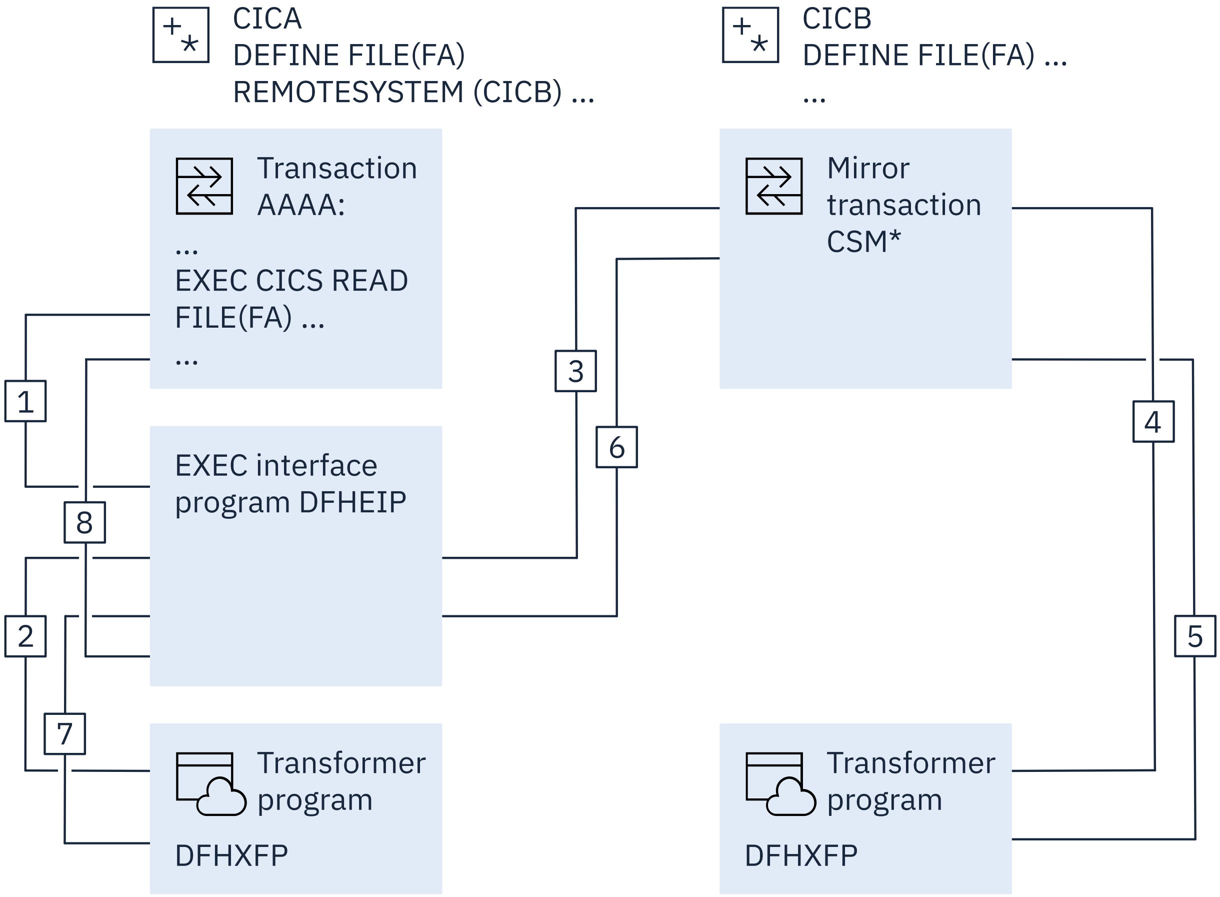


Figure 12. The transformer program and the mirror in function shipping

The sequence of events in Figure 12 on page 44 are as follows:

- In the requesting system (CICA in Figure 12 on page 44), the command-level EXEC interface program (for all except DL/I requests) determines that the requested resource is on another system (CICB in the example). It therefore calls the function-shipping transformer program to transform the request into a form suitable for transmission (in the example, line 2 indicates this call). The EXEC interface program then calls on the intercommunication component to send the transformed request to the appropriate connected system (line 3). For DL/I requests, part of this function is handled by CICS DL/I interface modules. For guidance about DL/I request processing, see Overview of Database Control (DBCTL).
- The first request to a specific remote system on behalf of a transaction causes the communication component in the local system to precede the formatted request with the appropriate mirror transaction identifier, in order to attach this transaction in the remote system. Thereafter, it keeps track of whether the mirror transaction stops, and reinvokes it as required.
- The mirror transaction uses the function-shipping transformer program to decode the formatted request (line 4 in Figure 12 on page 44). The mirror then runs the corresponding command. On completion of the command, the mirror transaction uses the transformer program to construct a formatted reply (line 5). The mirror transaction returns this formatted reply to the requesting system, CICA (line 6). On CICA, the reply is decoded, again using the transformer program (line 7), and used to complete the original request made by the application program (line 8).
- If the mirror transaction is not required to update any protected resources, and no previous request updated a protected resource in its system, the mirror transaction stops after sending its reply. However, if the request causes the mirror transaction to change or update a protected resource, or if the request is for any DL/I program specification block (PSB), it does not stop until the requesting application program issues a synchronization point (sync point) request or ends successfully. If a browse is in progress, the mirror transaction does not end until the browse is complete.
- When the application program issues a sync point request, or ends successfully, the intercommunication component sends a message to the mirror transaction that causes it also to issue a sync point request and stop. The successful sync point by the mirror transaction is indicated in a response sent back to the requesting system, which then completes its sync point processing, thereby committing changes to any protected resources. If DL/I requests have been received from another system, CICS issues a DL/I TERM request as a part of the processing resulting from a sync point request made by the application program and carried out by the mirror transaction.
- The application program can access protected or unprotected resources in any order, and is not affected by the location of protected resources. They might all be in remote systems, for example. When the application program accesses resources in more than one remote system, the intercommunication component invokes a mirror transaction in each system to run requests for the application program. Each mirror transaction follows the rules described previously for ending, and when the application program reaches a sync point, the intercommunication component exchanges sync point messages with any mirror transactions that have not yet ended. This situation is called the *multiple-mirror*.
- The mirror transaction uses the CICS command-level interface to run CICS requests, and the DL/I CALL or the EXEC DLI interface to run DL/I requests. The request is thus processed as for any other transaction and the requested resource is located in the appropriate resource table. If its entry defines the resource as remote, the mirror transaction's request is formatted for transmission and sent to another mirror transaction in the specified system. This situation is called a *chained-mirror*. To guard against possible threats to data integrity caused by session failures, you are recommended to avoid defining a connected system in which chained mirror requests occur, except when the requests involved do not access protected resources, or are inquiry-only requests.

Long-running mirror tasks for MRO

Normally, MRO mirror tasks are stopped as soon as possible, in the same way as described for ISC mirrors, to keep the number of active tasks to a minimum, and to avoid holding on to the session for long periods.

However, for some applications, it is more efficient to retain both the mirror task and the session until the next sync point, though this retention is not required for data integrity. For example, a transaction that issues many READ FILE requests to a remote system might be better served by a single mirror task, rather than by a separate mirror task for each request. In this way, you can reduce the overheads of allocating sessions on the sending side and attaching mirror tasks on the receiving side.

Mirror tasks that wait for the next sync point, even though they logically do not need to do so, are called *long-running mirrors*. They are applicable to MRO and IPIC links only. For MRO links, long-running mirror tasks are specified, on the system on which the mirror runs, by coding MROLRM=YES in the system initialization parameters. A long-running mirror is stopped by the next sync point (or RETURN) on the sending side.

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

Figure 14 on page 48 and Figure 15 on page 48 in "Function shipping examples" on page 47 show how the mirror acts for MROLRM=NO and MROLRM=YES, respectively.

An additional system initialization parameter, MROFSE=YES, specified on the front-end region, extends the retention of the mirror task and the session from the next sync point to the end of the task. To achieve maximum benefit, use MROFSE=YES with MROLRM=YES on the back-end region. However, MROFSE=YES still applies if the back-end region has MROLRM=NO, if requests are of the type that cause the mirror transaction to keep its inbound session.

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

MROFSE=YES gives a performance improvement only if most applications initiated from the front-end region have multiple sync points, and function shipping requests are issued between each sync point.

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

Long-running mirror tasks are also available over IPIC links. The lifetime of the mirror is specified using the MIRRORLIFE attribute on the IPCONN resource definition. See Long-running mirror tasks for IPIC.

The short-path transformer for MRO

CICS uses a special transformer program (DFHXFX) for function shipping over MRO links.

This *short-path transformer* optimizes the path length involved in the construction of the terminal input/output areas (TIOA) that are sent on an MRO session for function shipping. It optimizes the path length by using a private CICS format for the transformed request, rather than the architected format defined by SNA.

CICS uses DFHXFX for shipping file control, transient data, temporary storage, and interval control (asynchronous processing) requests. It is not used for DL/I requests. The shipped request always specifies the CICS mirror transaction, CSMI. Architected process names are not used.

Long-running mirror tasks for IPIC

Normally, IPIC mirror tasks are stopped as soon as possible, in the same way as described for ISC mirrors, to keep the number of active tasks to a minimum and to avoid holding on to the session for long periods.

However, for some applications, it is more efficient to retain both the mirror task and the session until the next sync point, though this rentention is not required for data integrity. For example, a transaction that issues many READ FILE requests to a remote system might be better served by a single mirror task, rather than by a separate mirror task for each request. In this way, you can reduce the overheads of allocating sessions on the sending side and attaching mirror tasks on the receiving side.

Mirror tasks that wait for the next sync point, or beyond the next sync point, even though they logically do not need to do so, are called *long-running mirrors*. They are applicable to MRO and IPIC links only. For IPIC links, the lifetime of the mirror is specified on the system on which the mirror runs by using the MIRRORLIFE attribute of the IPCONN on which the request is received. A long-running mirror for an

IPCONN specified with MIRRORLIFE(UOW) is stopped by the next sync point (or RETURN) on the sending side. A long-running mirror for an IPCONN specified with MIRRORLIFE(TASK) is stopped by the end of the task on the sending side.

For some applications, the performance benefits of using long-running mirrors can be significant. MIRRORLIFE(TASK) improves performance only if most applications that are initiated from the front-end region have multiple sync points and function shipping requests are issued between each sync point.

Specify MIRRORLIFE(TASK) or MIRRORLIFE(UOW) with caution, especially if distributed program link (DPL) requests with SYNCONRETURN or TRANSID are used.

Do not specify MIRRORLIFE(TASK) when long-running tasks might be used to function ship requests. The long-running tasks will retain the use of a SEND session for its entire duration and the SEND session will not be available for allocation to other tasks when it is no longer used. The MIRRORLIFE setting is not reflected in the lifetime of the mirror task until a file control, transient data queue (TDQ) or temporary storage queue (TSQ) request is function shipped.

Error handling and failure of the mirror transaction

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

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

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

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

However, in common with local execution (that is, when not using function shipping or distributed program link), the application program that issued the request that was function-shipped might attempt to handle the error or abend. The handle logic then issues an **EXEC CICS SYNCPOINT**, **SYNCPOINT ROLLBACK**, **RETURN**, or **ABEND** command. Attempting a **SYNCPOINT** or **RETURN**, (rather than a **SYNCPOINT ROLLBACK** or **ABEND**) despite being informed of the error or abend, results in an attempt to commit the client program's local resource updates and those performed by the server transaction before the error or abend was encountered.

If the mirror program cannot handle the error or abend encountered by the mirror transaction and this causes the termination and backout of the mirror transaction without sending a response to the client application, CICS forces the client program's transaction to back out. Any explicit sync point attempt fails and the local updates are backed out. This response also occurs if a problem is encountered with the communications link between the client and server tasks.

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

Function shipping examples

These examples illustrate the lifetime of the mirror transaction and the information that flows between the application and its mirror.

The examples contrast the action of the mirror transaction when accessing protected and unprotected resources on behalf of the application program, over MRO, ISC, or IPIC links, with and without MRO long-running mirror tasks.

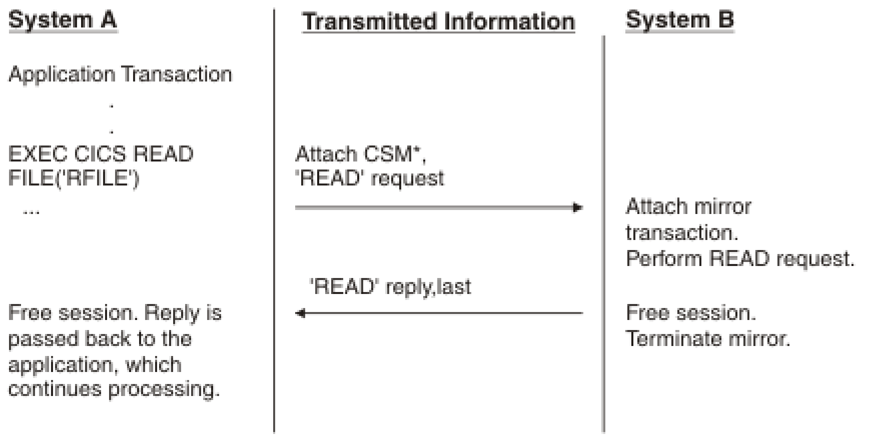


Figure 13. ISC function shipping: simple inquiry

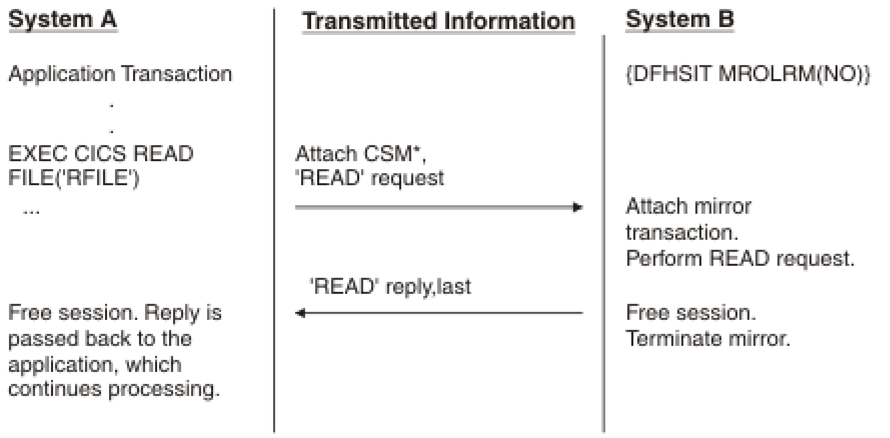


Figure 14. MRO or IPIC function shipping: simple inquiry

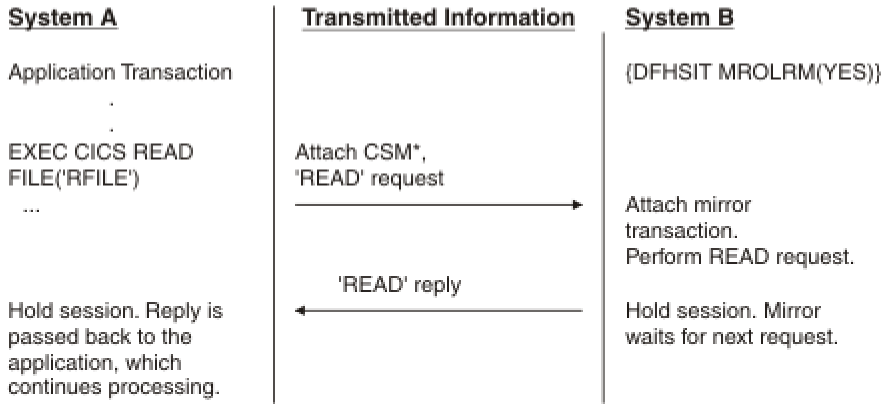


Figure 15. MRO or IPIC function shipping: simple inquiry

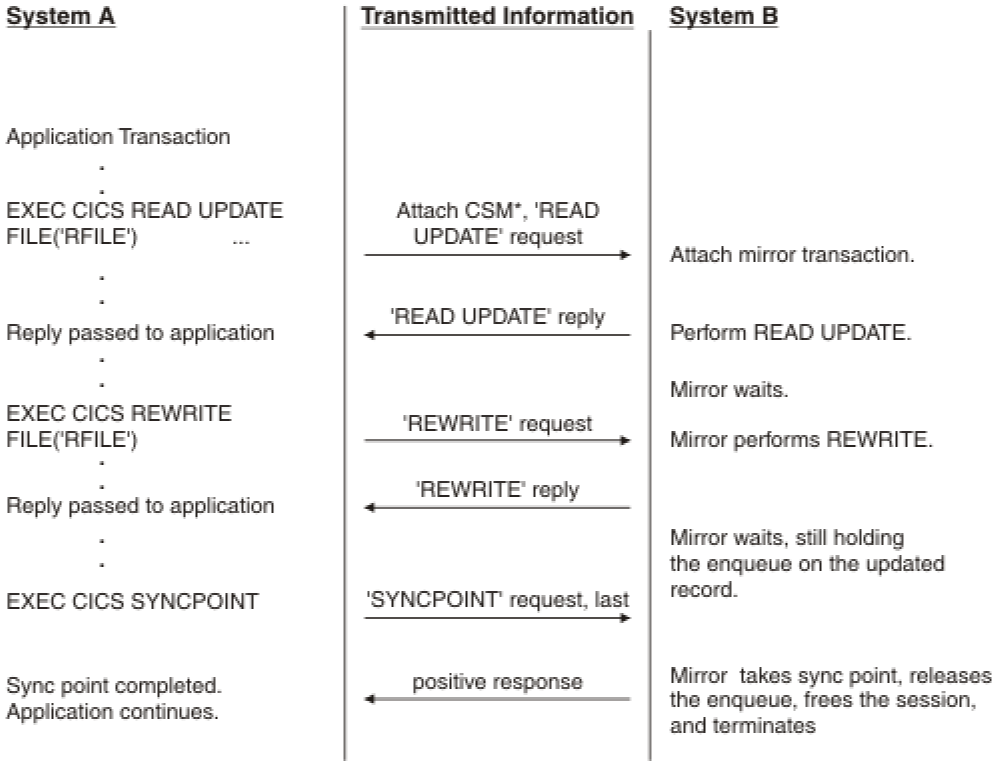


Figure 16. ISC, MRO, or IPIC function shipping: update

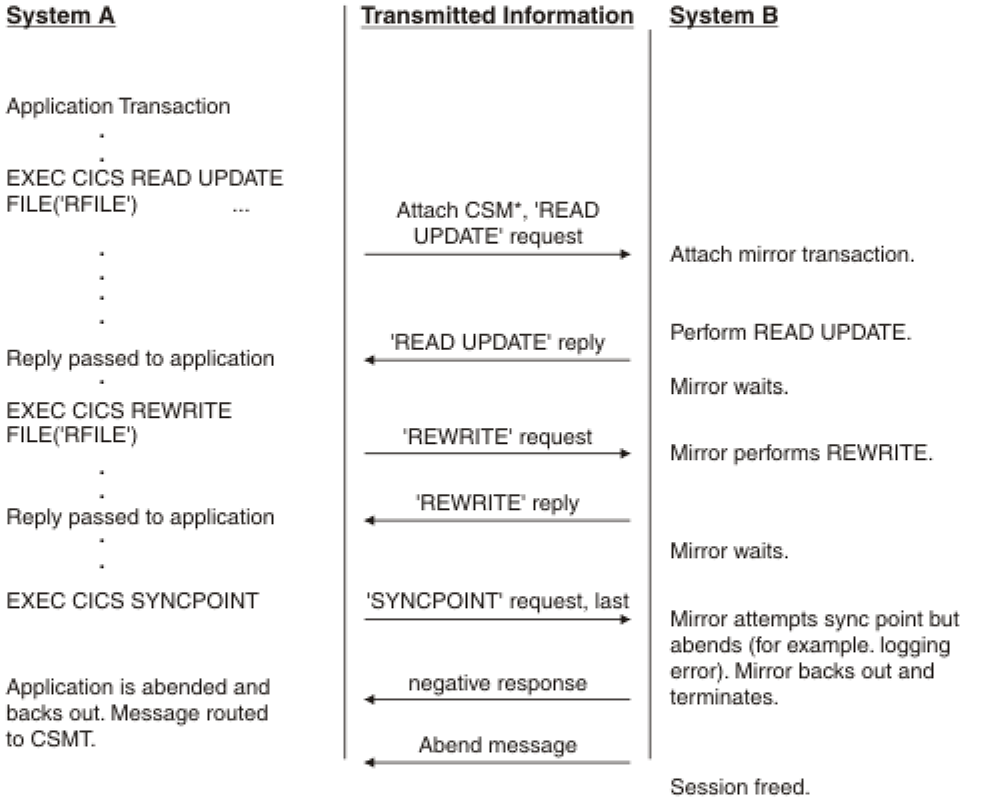


Figure 17. ISC, MRO, or IPIC function shipping: update with ABEND.

Figure 17 on page 49 is like Figure 16 on page 49, except that an abend occurs during sync point processing.

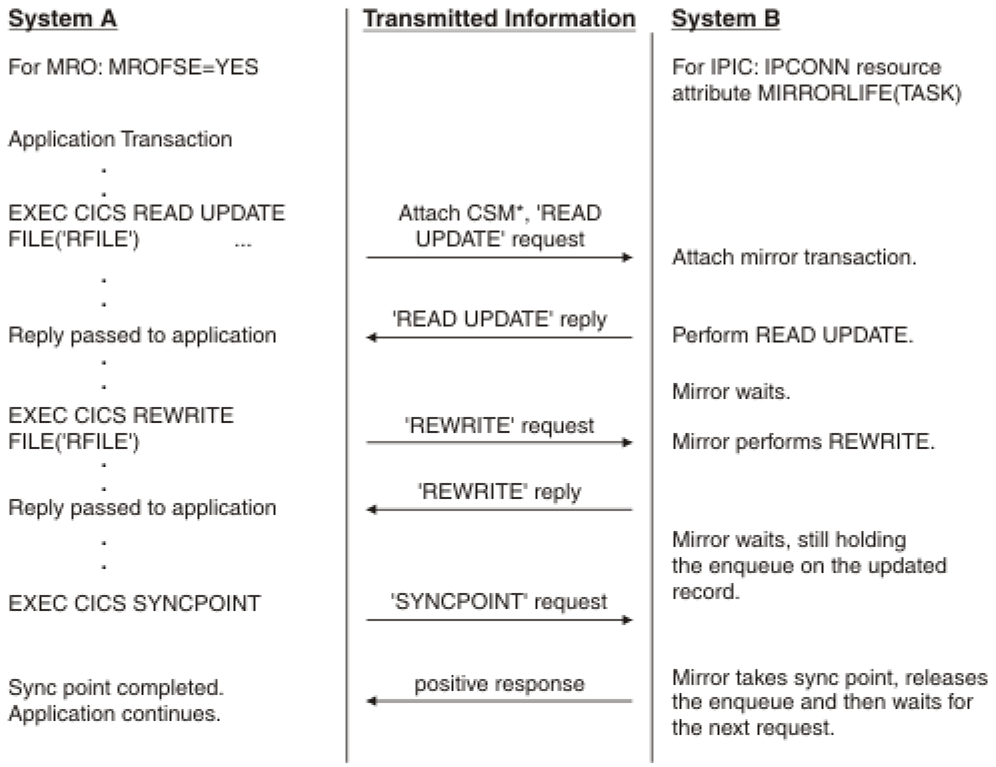

System A
Transmitted Information
System B
For MRO: MROFSE=YES
For IPIC: IPCONN resource attribute MIRRORLIFE(TASK)
Application Transaction
EXEC CICS READ UPDATE FILE('RFILE') ...
Attach CSM*, 'READ UPDATE' request
Attach mirror transaction.
'READ UPDATE' reply
Reply passed to application
Perform READ UPDATE.
Mirror waits.
EXEC CICS REWRITE FILE('RFILE')
'REWRITE' request
Mirror performs REWRITE.
'REWRITE' reply
Reply passed to application
Mirror waits, still holding the enqueue on the updated record.
EXEC CICS SYNCPOINT
'SYNCPOINT' request
positive response
Sync point completed. Application continues.
Mirror takes sync point, releases the enqueue and then waits for the next request.

Figure 18. MRO or IPIC function shipping: update using MROFSE or IPCONN MIRRORLIFE(TASK) to extend the life of the mirror transactions

Asynchronous processing

Asynchronous processing distributes the processing required by an application between intercommunicating systems. The processing is independent of the sessions on which requests are sent and replies are received.

This chapter contains the following topics:

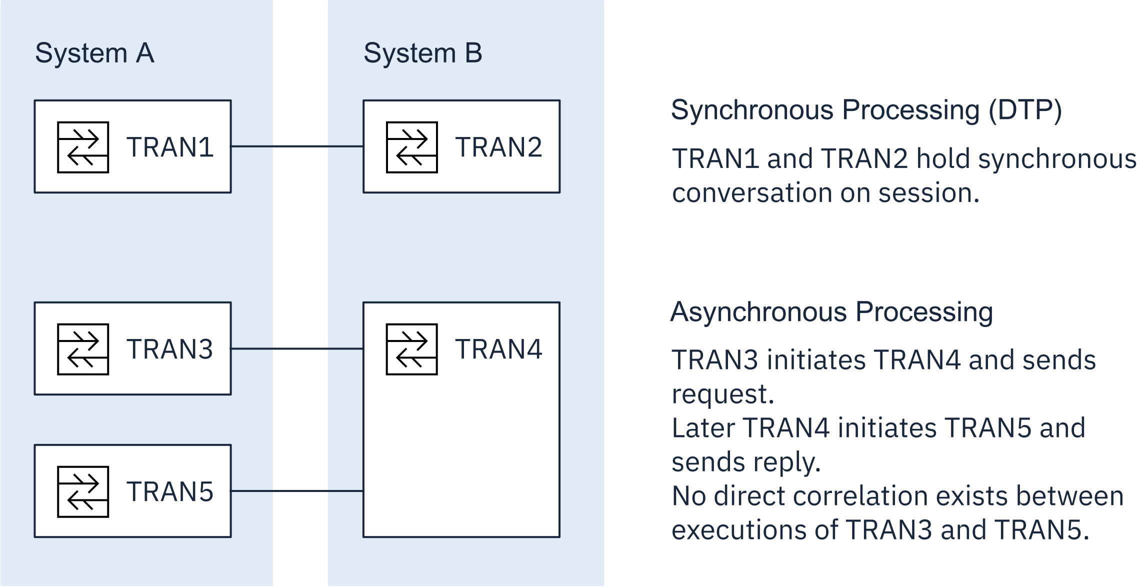
- "Overview of asynchronous processing" on page 50
- "Asynchronous processing methods" on page 51
- "Asynchronous processing using START and RETRIEVE commands" on page 52
- "System programming considerations" on page 57
- "Asynchronous processing examples" on page 58.

Overview of asynchronous processing

Asynchronous processing provides a means of distributing the processing that is required by an application between systems in an intercommunication environment. Unlike distributed transaction processing, however, the processing is **asynchronous**.

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

In asynchronous processing, the processing is independent of the sessions on which requests are sent and replies are received. No direct correlation can be made between a request and a reply, and no assumptions can be made about the timing of the reply. These differences are illustrated in Figure 19 on page 51.



Synchronous Processing (DTP)

TRAN1 and TRAN2 hold synchronous conversation on session.

Asynchronous Processing

TRAN3 initiates TRAN4 and sends request.
Later TRAN4 initiates TRAN5 and sends reply.
No direct correlation exists between executions of TRAN3 and TRAN5.

Figure 19. Synchronous and asynchronous processing compared

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

In general, asynchronous processing is applicable to any situation in which it is not necessary or desirable to tie up local resources while a remote request is being processed.

Asynchronous processing is not suitable for applications that involve synchronized changes to local and remote resources; for example, it cannot be used to process simultaneous linked updates to data split between two systems.

Asynchronous processing methods

In CICS, asynchronous processing can be done in one of two ways: by using the interval control commands START and RETRIEVE, or by using distributed transaction processing (DTP).

1. Asynchronous processing using the interval control commands START and RETRIEVE.

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

   If you use the START command for asynchronous processing, you can communicate only with systems that support the special protocol needed for function shipping; that is, CICS itself and IMS.

   A CICS transaction that is initiated by a remotely issued start request can use the RETRIEVE command to retrieve any data associated with the request. Data transfer is restricted to a single record passing from the initiating transaction to the transaction initiated.

2. Asynchronous processing using distributed transaction processing (DTP).

   This is a cross-system method and has no single-system equivalent. You can use it to initiate a transaction in a remote system that supports one of the DTP protocols.

When you use DTP to attach a remote transaction, you also allocate a session and start a conversation. This permits you to send data directly and, if you want, to receive data from the remote transaction. Your transaction design determines the format and volume of the data you exchange. For example, you can use repeated SEND commands to pass multirecord files.

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

The procedure to be followed by the two transactions while they are working together is determined by the application programming interface (API) for the protocol you are using. APPC is the preferred one, although you must use LUTYPE6.1 if you want to communicate with IMS. You might want to take advantage of the flexible data exchange facilities by employing this method across MRO links too.

Whatever protocol you decide to use, you must observe the rules it imposes. However short the conversation, during the time it is in progress, the processing is synchronous. In terms of command sequencing, error recovery, and syncpointing, it is full DTP.

In both forms of asynchronous processing (and also in synchronous processing), a CICS transaction can use the EXEC CICS ASSIGN STARTCODE command to determine how it was initiated.

CICS-to-IMS communication includes a special case of the DTP method described previously. Because it restricts data communication to one SEND LAST command answered by a single RECEIVE, this book refers to it elsewhere as the SEND/RECEIVE interface. The circumstances under which it is used are described in CICS-to-IMS applications.

Distributed transaction processing is described in "Distributed transaction processing" on page 109.

Asynchronous processing using START and RETRIEVE commands

The following interval control commands can be used for asynchronous processing.

- START
- CANCEL
- RETRIEVE.

For programming information about CICSinterval control, see Interval control. .

Starting and canceling remote transactions

The START and CANCEL commands are function shipped to the remote CICS or IMS system. If the remote system is CICS, the mirror transaction is started in the remote system to issue the START command on that system.

About this task

For asynchronous processing of threadsafe programs in a remote CICS system, performance is affected by the intercommunication method that you use for CICS-to-CICS communication. If you use IP interconnectivity (IPIC) over TCP/IP to connect the CICS systems, CICS uses an L8 open TCB whenever possible to run the mirror program used by the mirror transaction, so some TCB switching can be avoided. If you use MRO or ISC over SNA to connect the CICS systems, the mirror program does not run on an open TCB. The START and CANCEL commands are not threadsafe for any intercommunication method.

Procedure

- Use the interval control START command to schedule transactions asynchronously in remote CICS and IMS systems.
- For CICS-to-CICS communication, include time-control information on the shipped START command using the INTERVAL or TIME options.
  - A TIME specification is converted by CICS to a time interval, relative to the local clock, before the command is shipped. Because the ends of an intersystem link might be in different time zones, it

is typically better to think in terms of time intervals, rather than absolute times, for intersystem communication.

  - Note particularly that the time interval specified on a START command specifies the time at which the remote transaction is to be initiated, not the time at which the request is to be shipped to the remote system.

- You cannot specify time control for START commands sent to IMS systems. INTERVAL(0) must be specified or allowed to take the default value.
- You can cancel a START command shipped to a remote CICS system at any time up to its expiration time by shipping a CANCEL command to the same system.

  The particular START command has a unique identifier (REQID), which you can specify on the START command and on the associated CANCEL command. Any task that knows the identifier can issue the CANCEL command.

  For information about canceling dynamically-routed START commands, see "Canceling interval control requests" on page 92.
- Start requests for IMS transactions cannot be canceled after they have been issued, because you cannot specify time control for START commands sent to IMS systems.

Passing information with the START command

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

About this task

The information that can be specified is summarized in the following list:

- User data—specified in the FROM option.

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

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

  The QUEUE option cannot be used for CICS-to-IMS communication.
- The transaction and terminal names to be used for replies—specified in the RTRANSID and RTERMID options.

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

  For CICS-to-CICS communication, this is the name of a terminal that is to be associated with the remote transaction when it is initiated. It may be that the terminal is defined on the region that owns the remote transaction but is not owned by that region. If so, it is obtained by the automatic transaction initiation (ATI) facility of transaction routing. See "Traditional routing of transactions started by ATI" on page 72.

  The global user exits XICTENF and XALTENF can be coded to cover the case of the terminal that is *shippable* but not defined in the application-owning region. See "Shipping terminals for automatic transaction initiation" on page 74.

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

Passing a sysid or applid with the START command

If you have a transaction that can be started from several different systems, and which is required to issue a START command to the system that initiated it, you can arrange for all of the invoking transactions to send their local system sysid or applid as part of the user data in the START command.

About this task

An initiating transaction can obtain its local sysid by using an ASSIGN SYSID command, or its applid by using an ASSIGN APPLID command.

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

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

Improving performance of intersystem START requests

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

About this task

Where the connection between the two systems is via the z/OS Communications Server, this can result in considerably improved performance. The price paid for better performance is the inability of CICS to detect some types of error in the START command.

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

The transaction attached as a result of the terminal operator's inquiry issues an appropriate START command with the NOCHECK option, which causes a single message to be sent to the appropriate remote system to start, asynchronously, a transaction that makes the inquiry. The command should specify the operator's terminal identifier. The transaction attached to the operator's terminal can now terminate, leaving the terminal available for either receiving the answer or initiating another request.

The remote system performs the requested inquiry on its local database, then issues a start request for the originating system. This command passes back the requested data, together with the operator's terminal identifier. Again, only one message passes between the two systems. The transaction that is then started in the originating system must format the data and display it at the operator's terminal.

If a system or session fails, the terminal operator must reenter the inquiry, and be prepared to receive duplicate replies. To aid the operator, either a correlation field must be shipped with each request, or all replies must be self-describing.

An example of intercommunication using the NOCHECK option is given in Figure 21 on page 59.

The NOCHECK option is always required when shipping of the START command is queued pending the establishment of links with the remote system (see “Local queuing of START commands” on page 56), or if the request is being shipped to IMS.

Including start request delivery in a unit of work

The delivery of a start request to a remote system can be made part of a unit of work by specifying the PROTECT option on the **START** command.

About this task

The PROTECT option indicates that the remote transaction must not be scheduled until the local one successfully completes a synchronization point (syncpoint). (It can take the syncpoint either by issuing a **SYNCPOINT** command or by terminating normally.)

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

If the remote system is IMS, no message must cross the link between the **START** command and the syncpoint. Both PROTECT and NOCHECK must be specified for all IMS recoverable transactions.

Note: The PROTECT option of the **START** command behaves differently, depending on whether you route over MRO or APPC. If you route over MRO, the start happens immediately but when you route over APPC, the start is deferred until an explicit or implicit syncpoint is taken.

Deferred transmission of START requests with NOCHECK option for ISC links

For START commands with the NOCHECK option, whether you specify PROTECT, CICS can defer transmission of the request to the remote system for ISC links. For MRO links and IP interconnectivity (IPIC), START requests with NOCHECK are not deferred.

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

- The transaction issues a further START command or any function shipping request for the same system.
- The transaction issues a SYNCPOINT command.
- The transaction stops with an implicit sync point.

The first, or only, start request transmitted from a transaction to a remote system carries the begin-bracket indicator; the last, or only, request carries the end-bracket indicator. Also, if any of the start requests issued by the transaction specifies PROTECT, the last request in the unit of work (UOW) carries the sync point request indicator. Deferred sending allows the indicators to be added to the deferred data, and thus reduces the number of transmissions required.

Start requests are processed differently, if there are limitations because of protocol, connection, or receiving system:

- For both the APPC and LUTYPE6.1 protocols, if the first START with NOCHECK is followed by a second START with NOCHECK command, CICS transmits the first command and defers the second.
- For LUTYPE6.1 and 6.2 protocols, the sequence of requests is transmitted in a single SNA bracket and, if the remote system is CICS, all the requests are handled by the same mirror task.
- For MRO and IPIC connections, if the first START with NOCHECK is followed by a second START with NOCHECK command, CICS transmits both commands.
- For IMS, no message can cross the link between a START request and the following sync point. Therefore, you cannot send multiple START NOCHECK PROTECT requests to IMS. Each request must be followed by a SYNCPOINT command or by termination of the transaction. IP interconnectivity (IPIC) does not support requests to IMS.

Intersystem queuing

If the link to a remote region is established, but there are no free sessions available, function shipped EXEC CICS START requests used to schedule remote transactions may be queued in the issuing region.

Performance problems can occur if the queue becomes excessively long. This problem is described on page "Intersystem queuing" on page 42.

For guidance information about controlling intersystem queues, see Intersystem session queue management.

Local queuing of START commands

If a remote system is unavailable, either because it is not active or because a connection cannot be established, an attempt to function ship a START request to the remote system usually results in the SYSIDERR condition being returned to the application.

Provided that the remote system is directly connected to this CICS system, and that you specify the NOCHECK option on the START command, you can arrange for the request to be queued locally, and forwarded when the required link is in service.

You cannot cancel a START request while it remains on the local queue. The request can be canceled only when the required link is back in service, the request has been sent to the target region, and before the request is run.

A SYSIDERR condition is also returned when there *is* a connection to the remote system, but there are no sessions available and you have chosen not to queue the request in the issuing region. You can specify local queuing in two ways:

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

   For information about the LOCALQ option, see TRANSACTION attributes.
2. Use an XISLCLQ or XISQLCL global user exit program.

   XISLCLQ is invoked only for function-shipped **`EXEC CICS START NOCHECK`** commands, which are scheduled for a non-IPIC connection, when these conditions apply:

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

   XISQLCL is invoked for **`EXEC CICS START NOCHECK`** commands, which are scheduled for an IPIC connection, when these conditions apply:

   - The IPIC connection is not acquired.
   - A session is not available and CICS does not queue the request for a new session.

   If the connection resource is discarded, any requests that you have added to the local queue are lost.

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

   For programming information about the XISCONA, XISLCLQ, and XISQLCL global user exits, see Intersystem communication program exits, XISCONA, XISLCLQ, and XISQLCL.

Data retrieval by a started transaction

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

In accordance with the normal rules for CICS interval control, a start request for a particular transaction that carries both user data and a terminal identifier is queued if the transaction is already active and associated with the same terminal. During the waiting period, the data associated with the queued request can be accessed by the active transaction by using a further RETRIEVE command. This has the effect of canceling the queued start request.

Thus, it is possible to design transactions that can handle the data associated with multiple start requests. Typically, a long-running local transaction could be designed to accept multiple inquiries from a terminal and ship start requests to a remote system. From time to time, the transaction would issue RETRIEVE commands to receive the replies, the absence of further replies being indicated by the ENDDATA condition.

The WAIT option of the RETRIEVE command can be used to put the transaction into a wait state pending the arrival of the next start request from the remote system. If this option is used in a task attached to an APPC device, CICS does not suspend the task, but instead raises the ENDDATA condition if no data is currently available. However, for tasks attached to non-APPC devices, you must make sure that your transaction does not get into a permanent wait state in the absence of further start requests.

Important:

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

For information about the ROUTABLE option of the START command, see “Routing transactions invoked by START commands” on page 81.

Terminal acquisition by a remotely-initiated CICS transaction

When a CICS transaction is started by a start request that names a terminal (TERMID), CICS makes the terminal available to the transaction as its principal facility.

It makes no difference whether the start request was issued by a user transaction in the local CICS system or was received from a remote system and issued by the mirror transaction.

Starting transactions with ISC or MRO sessions

You can name a system, rather than a terminal, in the TERMID option of the START command.

About this task

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

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

System programming considerations

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

- A link to a remote system must be defined.
- Remote transactions that are to be initiated by start requests must be defined as remote resources to the local CICS system. This is not necessary, however, for transactions that are initiated only by START commands that name the remote system explicitly in the SYSID option.
- If the QUEUE option is used, the named queue must be defined on the system to which the start request is shipped. The queue can be either a local or a remote resource on that system.
- If a START request names a “reply” transaction, that transaction must be defined on the system to which the start request is shipped.

Asynchronous processing examples

These examples show you how remote transactions are initiated over MRO, ISC, and IPIC connections.

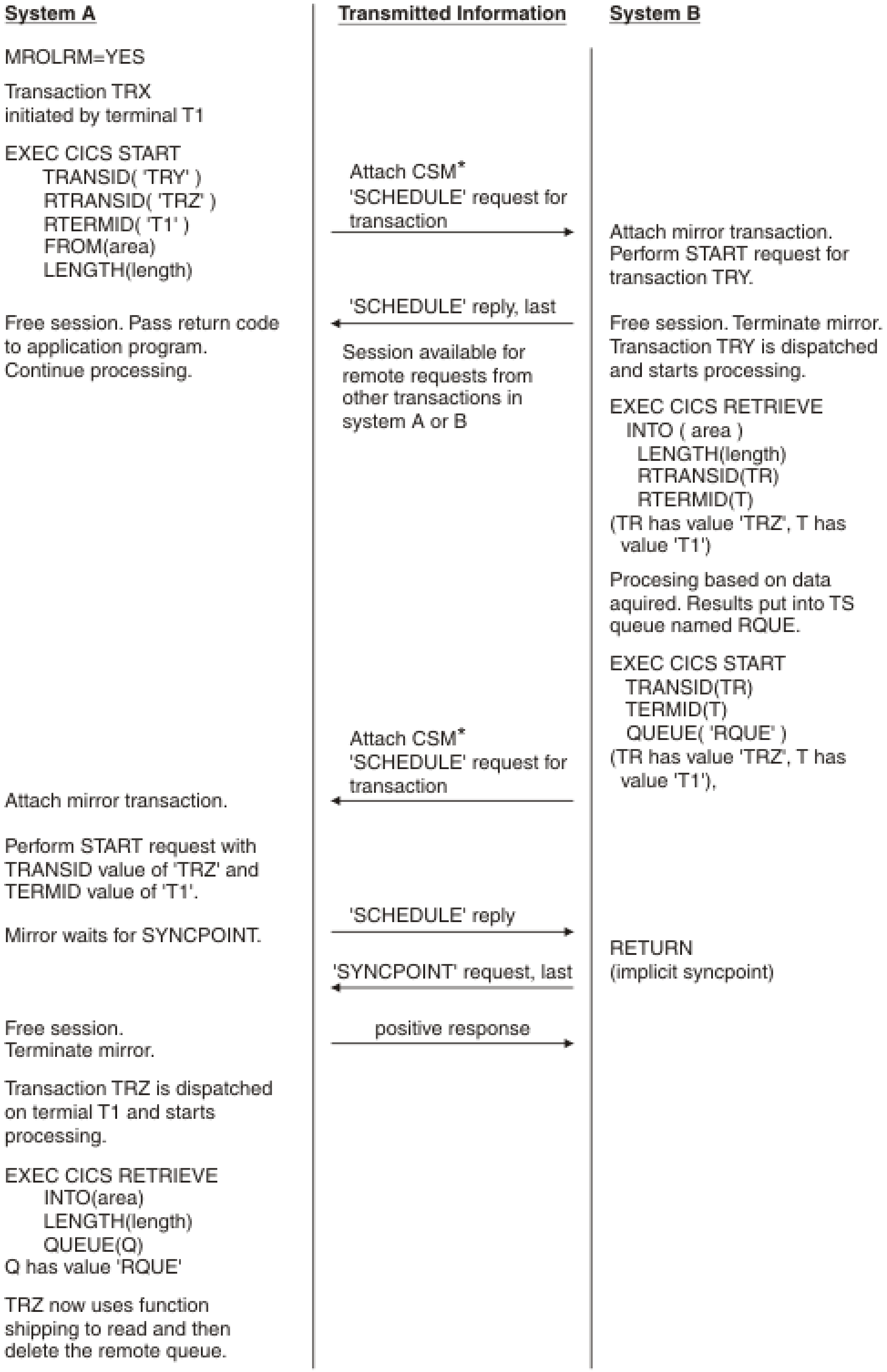
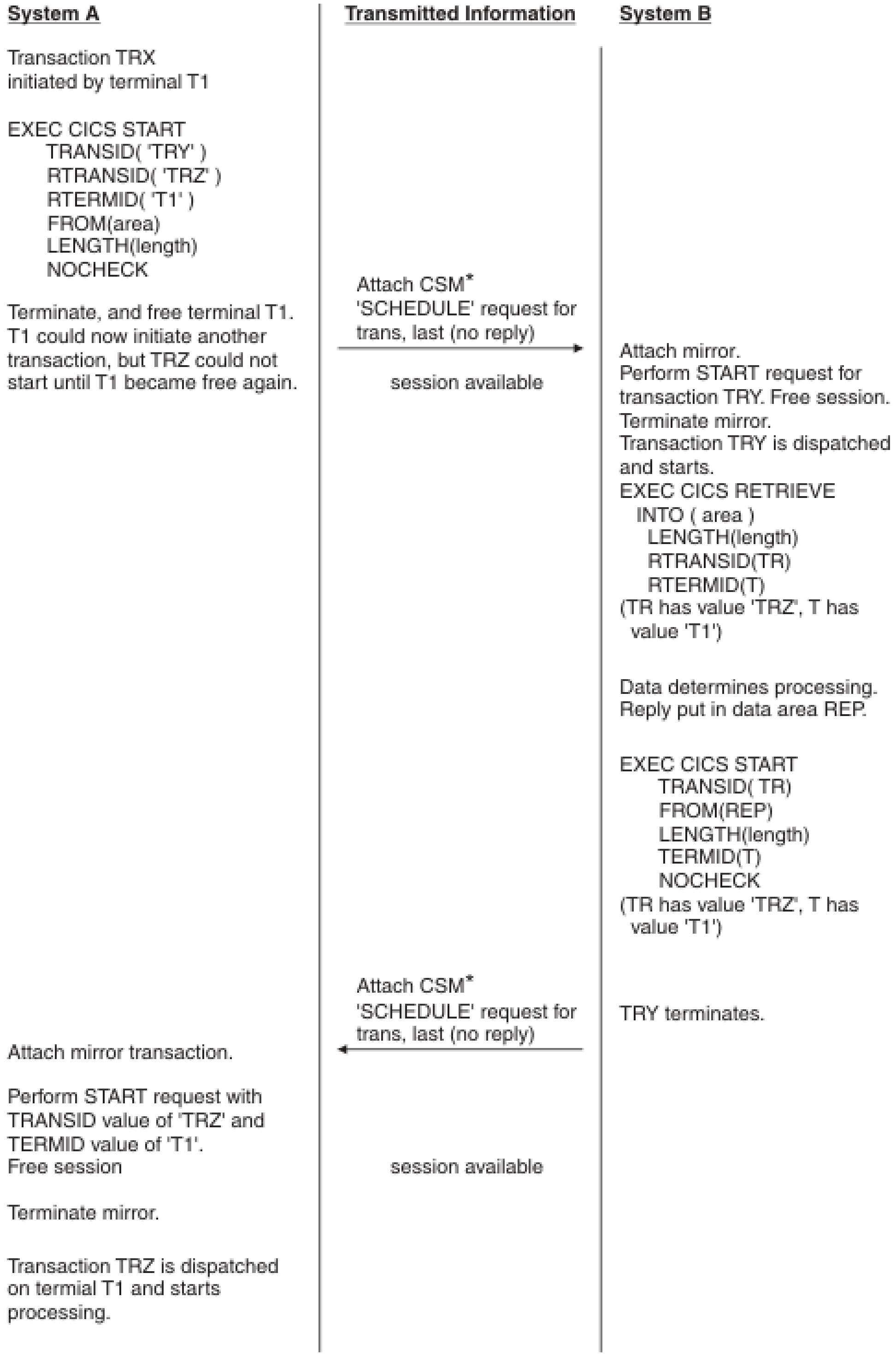


Figure 20. Asynchronous processing—remote transaction initiation



System A

Transaction TRX
initiated by terminal T1

```
EXEC CICS START
      TRANSID( 'TRY' )
      RTRANSID( 'TRZ' )
      RTERMID( 'T1' )
      FROM(area)
      LENGTH(length)
      NOCHECK
```

Terminate, and free terminal T1. T1 could now initiate another transaction, but TRZ could not start until T1 became free again.

Attach mirror transaction.

Perform START request with TRANSID value of 'TRZ' and TERMID value of 'T1'.
Free session

Terminate mirror.

Transaction TRZ is dispatched on termial T1 and starts processing.

Transmitted Information

Attach CSM*
'SCHEDULE' request for trans, last (no reply)

session available

Attach CSM*
'SCHEDULE' request for trans, last (no reply)

session available

System B

Attach mirror.
Perform START request for transaction TRY. Free session.
Terminate mirror.
Transaction TRY is dispatched and starts.

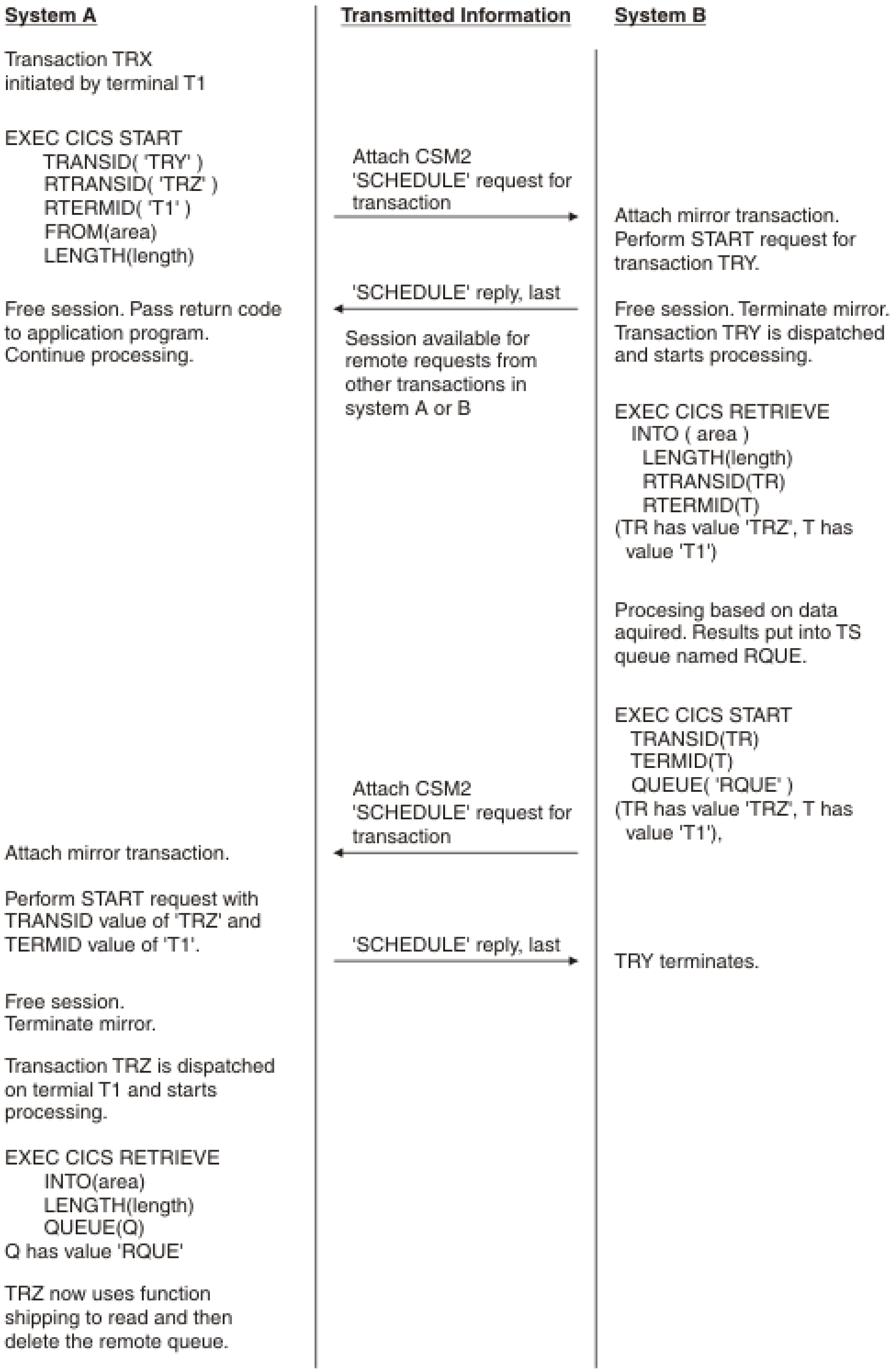
```
EXEC CICS RETRIEVE
   INTO ( area )
     LENGTH(length)
     RTRANSID(TR)
     RTERMID(T)
(TR has value 'TRZ', T has
  value 'T1')
```

Data determines processing.
Reply put in data area REP.

```
EXEC CICS START
      TRANSID( TR)
      FROM(REP)
      LENGTH(length)
      TERMID(T)
      NOCHECK
(TR has value 'TRZ', T has
  value 'T1')
```

TRY terminates.

Figure 21. Asynchronous processing—remote transaction initiation using NOCHECK



Transaction TRX
initiated by terminal T1

```
EXEC CICS START
      TRANSID( 'TRY' )
      RTRANSID( 'TRZ' )
      RTERMID( 'T1' )
      FROM(area)
      LENGTH(length)
```

Free session. Pass return code to application program. Continue processing.

Attach mirror transaction.

Perform START request with TRANSID value of 'TRZ' and TERMID value of 'T1'.

Free session.
Terminate mirror.

Transaction TRZ is dispatched on termial T1 and starts processing.

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

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

Attach CSM2 'SCHEDULE' request for transaction

'SCHEDULE' reply, last

Session available for remote requests from other transactions in system A or B

Attach CSM2 'SCHEDULE' request for transaction

'SCHEDULE' reply, last

Attach mirror transaction. Perform START request for transaction TRY.

Free session. Terminate mirror. Transaction TRY is dispatched and starts processing.

```
EXEC CICS RETRIEVE
  INTO ( area )
    LENGTH(length)
    RTRANSID(TR)
    RTERMID(T)
(TR has value 'TRZ', T has
 value 'T1')
```

Procesing based on data aquired. Results put into TS queue named RQUE.

```
EXEC CICS START
  TRANSID(TR)
  TERMID(T)
  QUEUE( 'RQUE' )
(TR has value 'TRZ', T has
 value 'T1'),
```

TRY terminates.

Figure 22. Asynchronous processing—remote transaction initiation

CICS dynamic routing

This section is an overview of the CICS dynamic routing interface.

The information it contains is relevant to both "CICS transaction routing" on page 67 and "CICS distributed program link" on page 98.

What is dynamic routing?

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

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

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

What requests can be dynamically routed?

All the following requests can be dynamically routed:

- Transactions started from terminals
- Transactions invoked by a subset of **EXEC CICS START** commands
- CICS-to-CICS distributed program link (DPL) requests
- Program-link requests received from outside CICS; for example, External Call Interface (ECI) calls received from CICS clients
- CICS business transaction services (BTS) processes and activities
- Bridge 3270 transactions
- CICS web service requests

What is required for dynamic routing?

Some further definitions are required:

Requesting region
The region in which a transaction or other request is issued. Here are some examples a requesting region:

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

Routing region
The region in which the routing program is invoked for route selection. With one exception, the requesting region and the routing region are always the same region. The exception is terminal-related START commands. A terminal-related START command is always executed in the terminal-owning region, the requesting region and the routing region may or may not be the same. (This is fully explained in “Routing transactions invoked by START commands” on page 81.) The routing region is always the TOR.

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

Two routing models

There are two possible dynamic routing models.

The hub model

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

Figure 23 on page 63 shows a hub routing model. A terminal-owning region (TOR) is connected to four application-owning regions (AORs). A dynamic routing program runs in the TOR and balances transaction requests across the AORs. The TOR is both the requesting region and the routing region. Each of the AORs is a possible target region.

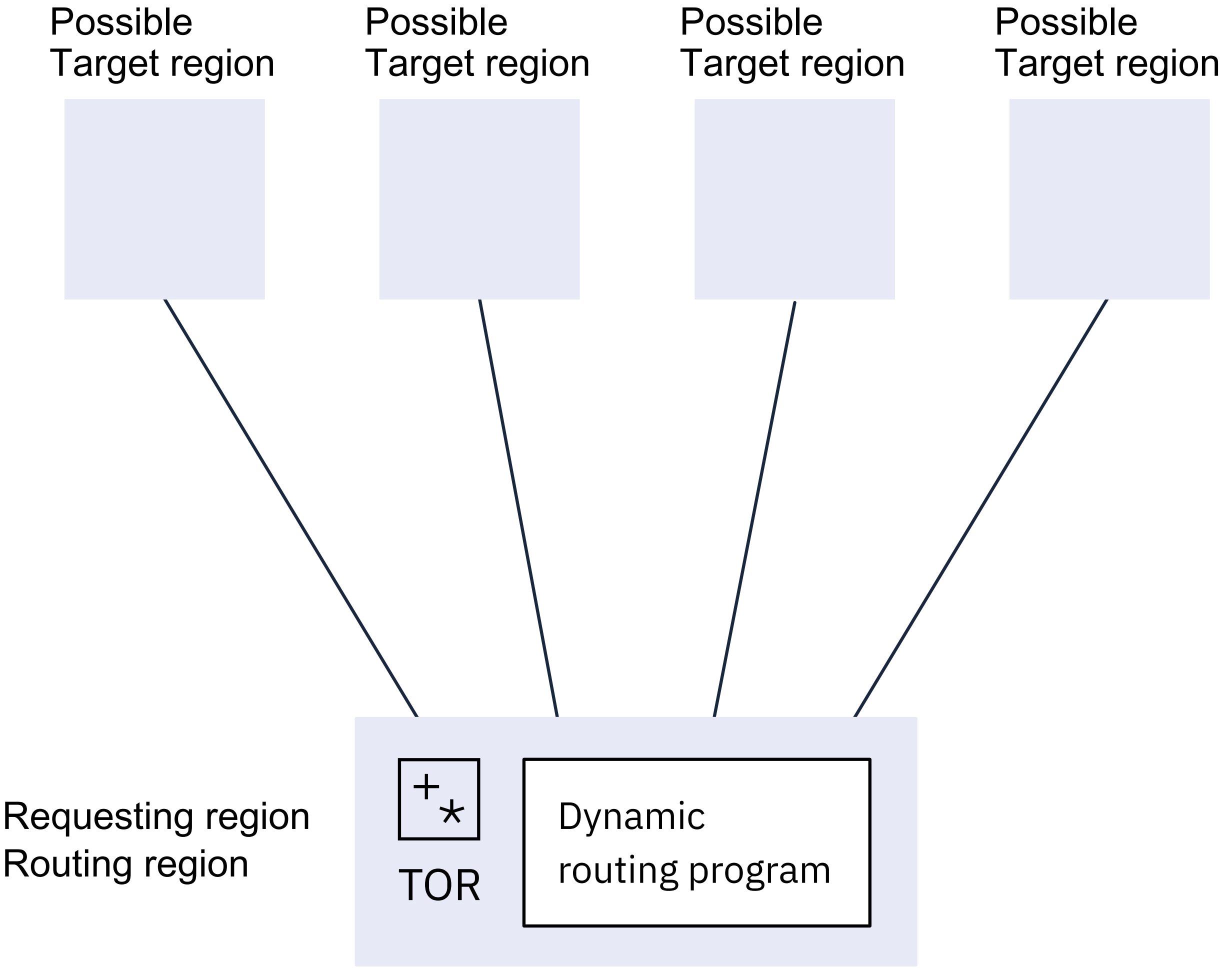


Figure 23. Dynamic routing using a hub routing model

The hub model applies to the routing of the following:

- Transactions started from terminals
- Transactions started by terminal-related **START** commands
- Program-link requests received from outside CICS

  The receiving region acts as a hub or TOR because it routes the requests among a set of back-end server regions.
- Bridge 3270 requests
- CICS-to-CICS DPL requests

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

Advantage of the hub model

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

Disadvantages of the hub model

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

The distributed model

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

Figure 24 on page 65 shows a distributed routing model. Four CICS regions are connected to each other. A distributed routing program runs in each region. Each region may be a requesting region, routing region, or target region.

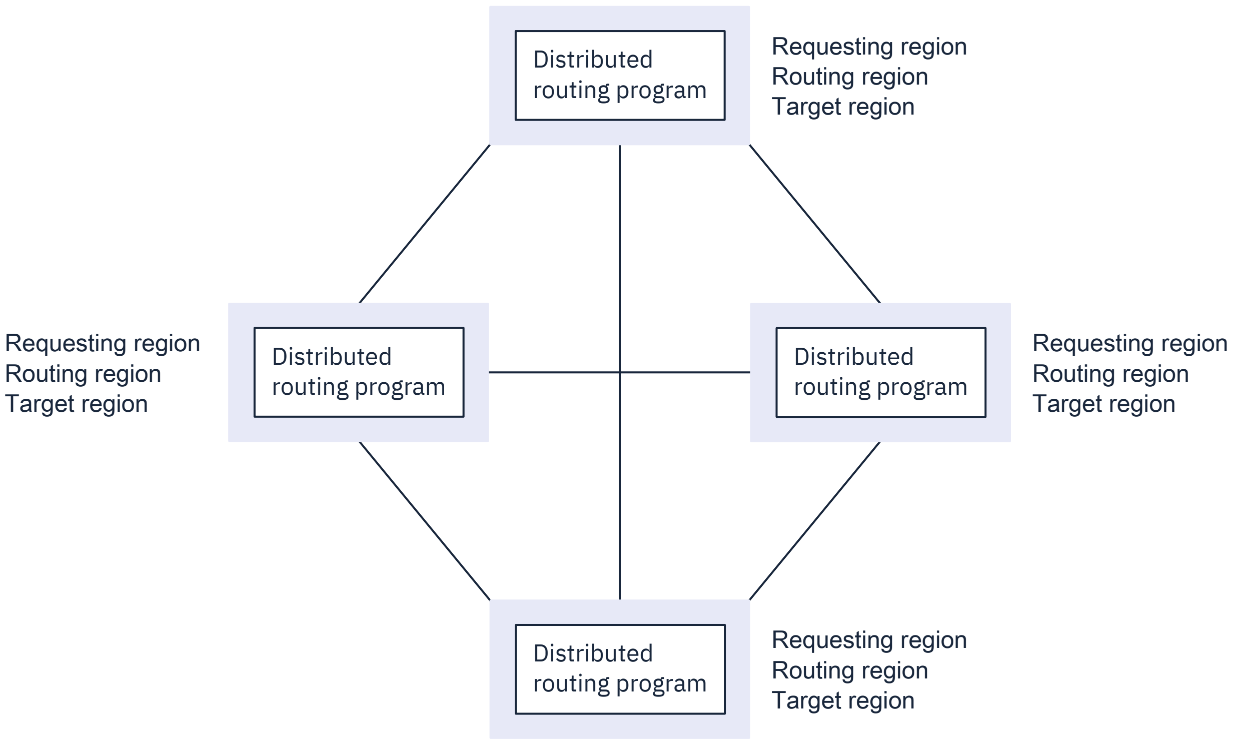


Figure 24. Dynamic routing using a distributed routing model

The distributed model applies to the routing of the following:

- CICS business transaction services processes and activities
- Non-terminal-related **START** requests
- CICS web service requests

Advantage of the distributed model

There is no single point-of-failure.

Disadvantages of the distributed model

- Compared to the hub model, there are a great many inter-region connections to maintain.
- You may have problems with distributed data. For example, any data used to make routing decisions must be available to all the regions. (CICSPlex SM solves this problem by using dataspaces.)

Two routing programs

CICS provides two user-replaceable programs for dynamic routing: the dynamic routing program and the distributed routing program. If you are using CICSPlex SM to manage your CICS environment, you can use the EYU9XLOP routing program instead.

You can use the dynamic routing program, DFHDYP, to route the following requests:

- Transactions started from terminals
- Transactions started by terminal-related START commands
- CICS-to-CICS DPL requests
- Program-link requests received from outside CICS
- Bridge 3270 requests

You can use the distributed routing program, DFHDSRP, to route the following requests:

- CICS business transaction services processes and activities
- Non-terminal-related START requests
- CICS web service requests

The two routing programs are specified on different system initialization parameters. You specify the name of the dynamic routing program on the **DTRPGM** system initialization parameter. You specify the name of the distributed routing program on the **DSRTPGM** system initialization parameter. The distributed routing program must be specified in the routing and target CICS regions.

The programs are passed the same communications area. However, certain fields that are meaningful to one program are not meaningful to the other. The programs are also called at similar points; for example, for route selection, route selection error, and optionally at termination of the routed transaction or program-link request.

You have flexibility to use these programs in any of the following ways:

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

The dynamic and distributed routing programs are different in two important ways:

- The dynamic routing program and the distributed routing program are called if the resource (the transaction or program) is defined as DYNAMIC(YES). However, in the case of BTS activities that are run asynchronously, the distributed routing program is called even if the associated transaction is defined as DYNAMIC(NO). In this situation, the distributed routing program cannot route the request, but it can monitor the effect of the request on workloads, or perform other activities. This difference means that you can use the distributed routing program to monitor the effect of statically-routed requests on the relative workloads of the target regions.

- The dynamic routing program uses the hierarchical *hub* routing model, where one routing program controls access to resources on several target regions. The routing program that is called at termination of a routed request is the same program that was invoked for route selection.

  The distributed routing program uses the distributed model, which is a peer-to-peer system; the routing program itself is distributed. The routing program that is invoked at initiation or termination of a routed transaction is not the same program that was invoked for route selection. It is the routing program on the target region. You must ensure that a distributed routing program is specified in all the target regions in addition to the routing region.

CICS transaction routing

CICS transaction routing allows terminals connected to one CICS system to run transactions in another CICS system.

Overview of transaction routing

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

Figure 25 on page 68 shows a terminal connected to one CICS system running with a user transaction in another CICS system. Communication between the terminal and the user transaction is handled by a CICS-supplied transaction called the *relay transaction*.

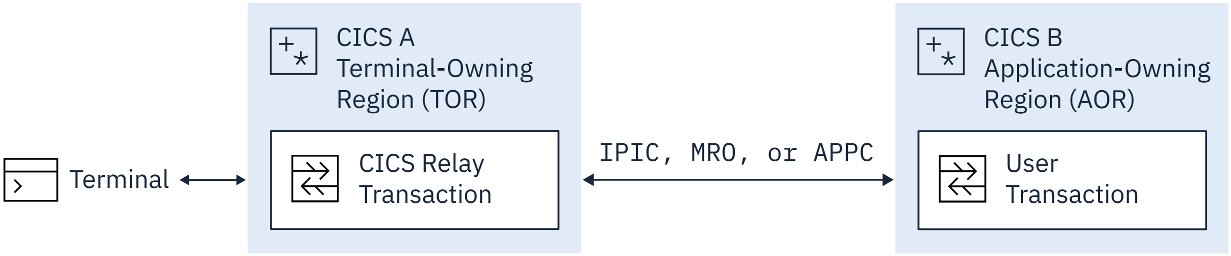


Figure 25. The elements of transaction routing

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

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

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

- LUTYPE6.1 connections and sessions
- MRO connections and sessions
- EXCI connections and sessions
- IBM 7770 or 2260 terminals
- Pooled 3600 or 3650 pipeline logical units
- MVS system consoles

The user transaction can use the terminal control, BMS, or batch data interchange facilities of CICS to communicate with the terminal, as appropriate for the terminal or session type. Mapping and data interchange functions are performed in the application-owning region. BMS paging operations are performed in the terminal-owning region.

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

How to initiate transaction routing

Transaction routing can be initiated in three ways:

1. A request to start a transaction can arrive from a terminal connected to the TOR. On the basis of an installed resource definition for the transaction, and possibly on decisions made in a user-written dynamic routing program, the request is routed to an appropriate AOR, and the transaction runs as if the terminal were attached to the same region.
2. A transaction can be started by automatic transaction initiation (ATI) and can acquire a terminal that is owned by another CICS system. The two methods of routing transactions started by ATI are described in the following topics:
   - "Traditional routing of transactions started by ATI" on page 72
   - "Routing transactions invoked by START commands" on page 81
3. A transaction can issue an **ALLOCATE** command to obtain a session to an APPC terminal or connection that is owned by another system.

In addition to these methods, CICS provides a special transaction (CRTE) that can be used for the occasional invocation of transactions in other systems. See "Using the routing transaction, CRTE" on page 97.

Terminal-initiated transaction routing

When a request to start a transaction arrives at a CICS TOR, the TOR must find out on which system the transaction is to run. It does this by examining the installed transaction definition; in particular, the values of the `DYNAMIC` and `REMOTESYSTEM` options (see Defining transactions for transaction routing). Transaction routing can be either **static** or **dynamic**, depending upon the value of the `DYNAMIC` option.

Static transaction routing

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

Dynamic transaction routing

Dynamic transaction routing occurs when DYNAMIC(YES) is specified in the transaction definition. It means that you want the chance to route the terminal data to an alternative transaction at the time the defined transaction is invoked. CICS manages this by allowing a user-replaceable program, called the dynamic routing program, to intercept the terminal input data and specify that it be redirected to any transaction and system.

Dynamic transaction routing

If DYNAMIC(YES) is specified in the transaction definition, the terminal data is allowed to be routed to an alternative transaction at the time the defined transaction is invoked. CICS manages this by allowing a user-replaceable program, called the *dynamic routing program*, to intercept the terminal input data and specify that it be redirected to any transaction and system.

The default dynamic routing program, supplied with CICS, is named DFHDYP. You can modify the supplied program, or replace it with one that you write yourself. You can also use the **DTRPGM** system initialization parameter to specify the name of the program that is invoked for dynamic routing, if you want to name your program something other than DFHDYP. For programming information about user-replaceable programs in general, and about DFHDYP in particular, see Writing a dynamic routing program.

Dynamic routing models:

Dynamic routing of terminal-initiated transactions uses the *hub* routing model.

Using CICSPlex SM-based dynamic routing

Without CICSPlex SM, to take advantage of dynamic transaction routing, you must either customize and implement the CICS supplied dynamic transaction routing program DFHDYP or DFHDSRP, or write your own. However, if you use CICSPlex SM to manage your CICSplex, you need not do so. CICSPlex SM provides a dynamic routing program that supports both workload routing and workload separation. All that you need do is to tell CICSPlex SM which TORs and AORs in the CICSplex can participate in dynamic transaction routing, and define any affinities that govern the AORs to which particular transactions must be routed.

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

The output from the CICS Interdependency Analyzer can be used directly by CICSPlex SM.

When your routing program is invoked

CICS invokes the dynamic routing program in the following situations.

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

  Note:

  1. If a transaction definition is not found, CICS uses the common transaction definition specified on the **DTRTRAN** system initialization parameter. See Using a single transaction definition in the TOR.
  2. If the transaction is defined as DYNAMIC(YES) in the target region, as well as in the routing region (TOR), the dynamic routing program is invoked, for routing, in the target region, as well as in the TOR. Thus, for 3270 transactions, it is possible to "daisy-chain" routed requests from one region to another. Take care that this does not occur unintentionally.

  If the transaction was initiated from a terminal, the dynamic routing program can route the request; see “Overview of transaction routing” on page 67.

  If the transaction was initiated by an **EXEC CICS START** command, the routing program may or may not be able to route the request; see “Routing transactions invoked by START commands” on page 81.
- If an error occurs in route selection.
- At the end of a routed transaction, if the initial invocation requests re-invocation at termination.

- If a routed transaction abends, if the initial invocation requests re-invocation at termination.
- For routing of DPL requests, at all the points described in “Dynamically routing DPL requests” on page 103.

Information passed to your routing program

Parameters are passed in a communications area between CICS and the dynamic routing program.

The program might change some of these parameters to influence subsequent CICS action. The parameters include:

- The reason for the current invocation.
- Error information.
- The sysid of the target system. Initially, the sysid specified on the REMOTESYSTEM option of the installed transaction definition. If no sysid was specified, the sysid passed is that of the local system.

  Use a single, common definition for all remote transactions that are to be dynamically routed. See Using a single transaction definition in the TOR.
- The name of the target transaction. Initially, the name specified on the REMOTENAME option for the installed transaction definition. If no name was specified, the name passed is the local name.
- The address of a buffer containing a copy of the data in the terminal input/output area (TIOA).
- The netname of the target system. Initially, the netname corresponds to the sysid specified on the REMOTESYSTEM option of the installed transaction definition.
- The address of the target transaction's communications area. If you are using channels and containers and you have defined a DFHROUTE container, DFHROUTE is used for the address.
- A user area.

Using your dynamic routing program

You can use dynamic transaction routing to make transaction routing decisions based on the input to the transaction, available CICS systems, relative loading of the available systems, and similar factors. However, a routing program can perform other functions, besides redirecting transaction requests.

Your dynamic routing program could be used for these purposes:

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

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

For programming information about writing a dynamic transaction routing program, see Writing a dynamic routing program.

The CICS Interdependency Analyzer

CICS transactions use many techniques to pass information between one another, and to synchronize activity between themselves.

Some of these techniques require the transactions exchanging data to execute in the same CICS region, and therefore impose restrictions on the dynamic routing of the transactions. If you are using dynamic transaction routing for workload balancing purposes (where equivalent transactions reside on multiple systems), your routing program must be aware of transactions that are dependent on each other (that is, that contain *affinities*) so that it can route them consistently.

If you are planning to create a dynamic transaction routing environment, consisting perhaps of a mixture of CICS Transaction Server for z/OS, Version 5 Release 6 and earlier systems, you may find the CICS Interdependency Analyzer useful. It can be used to identify the causes of inter-transaction affinities in CICS Transaction Server for z/OS regions.

For more information about this utility, see CICS Interdependency Analyzer for z/OS Overview.

For further information about transaction affinities, see Affinity.

Traditional routing of transactions started by ATI

You can use the traditional method of routing transactions that are started by automatic transaction initiation (ATI) for transactions where you cannot use the enhanced method.

This method of routing is superseded by the *enhanced method* (see "Routing transactions invoked by START commands" on page 81). Use the enhanced method wherever possible. However, for the following transactions, you must use the traditional method:

- Transactions invoked by the trigger-level on a transient data queue
- Transactions that are invoked by **EXEC CICS START** commands but do not meet the requirements, as explained in "How to route transactions started by terminal-related START commands" on page 82, to use the enhanced method

Automatic transaction initiation is the process where an internal transaction request in a CICS system or systems network leads to the scheduling of the transaction. ATI requests result from the following:

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

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

CICS transaction routing allows an ATI request for a transaction owned by a specific CICS system to name a terminal that is owned by another, connected system. For example, in Figure 26 on page 73, an application in AOR1 issues a START request for transaction TRAA to be attached to terminal PRT1.

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

The user transaction in the AOR is then accessed in the manner described for terminal-initiated transaction routing. Associated with the request is an automatic initiate descriptor (AID) that specifies the names of the remote transaction (TRAA) and system (AOR1).

For static transaction routing, the terminal-owning region (TOR1) must find a transaction definition that specifies REMOTESYSTEM(AOR1) and REMOTENAME(TRAA). If the TOR cannot find the correct definition, the request fails.

For dynamic transaction routing using the traditional method, when DYNAMIC(YES) is coded on the transaction definition, the dynamic routing program is invoked but cannot reroute the request, because the remote system name is taken from the AID. To find out how to use the ROUTABLE option of the transaction definition to specify enhanced routing, see "Routing transactions invoked by START commands" on page 81.

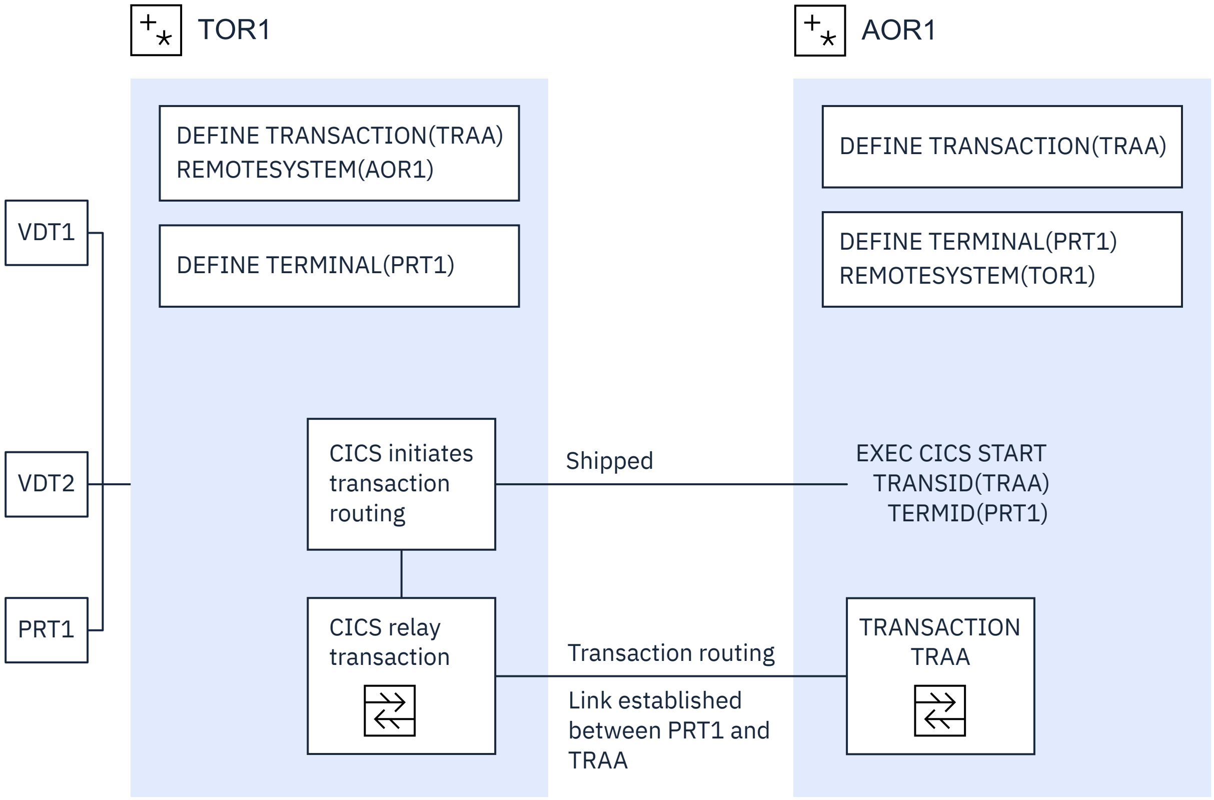

TOR1
AOR1
DEFINE TRANSACTION(TRAA) REMOTESYSTEM(AOR1)
DEFINE TERMINAL(PRT1)
DEFINE TRANSACTION(TRAA)
DEFINE TERMINAL(PRT1) REMOTESYSTEM(TOR1)
VDT1
VDT2
PRT1
CICS initiates transaction routing
Shipped
EXEC CICS START TRANSID(TRAA) TERMID(PRT1)
CICS relay transaction
Transaction routing
Link established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

Figure 26. ATI-initiated transaction routing

ATI requests are queued in the AOR if the link to the terminal-owning region is not available, and subsequently in the TOR if the terminal is not available.

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

In the AOR, the normal rules for ATI apply. The transaction can be initiated from a transient data queue when the trigger level is reached, or on expiry of an interval control start request. For transient data initiation, the transient data queue must be in the same system as the transaction. Transaction routing does not enable transient data queue entries to initiate remote transactions.

Shipping terminals for automatic transaction initiation

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

For example:

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

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

Terminal-not-known condition

The terminal-not-known condition frequently occurs because a terminal-related START command is issued in the terminal-owning region and function-shipped to the application-owning region, where the terminal is not yet defined.

Important:

If you can use the enhanced routing method described in “Routing transactions invoked by START commands” on page 81, a START command issued in a TOR is not function-shipped to the AOR; thus the terminal-not-known condition does not occur.

To ensure correct functioning of cross-region ATI, you could define your terminals to all the systems on the network that need to use them. However, you cannot do this if you are using autoinstall. See Autoinstall. Autoinstalled terminals are unknown to the system until they log on, and you rely on CICS to ship terminal definitions to all the systems where they are needed. (See Shipping terminal and connection definitions.) This works when routing from a terminal to a remote system, but there are cases where a system cannot process an ATI request, because it has not been told the location of the associated terminal.

The example shown in Figure 27 on page 76 should make this clear:

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

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

4. Because X1 is defined as a remote transaction owned by CICB, CICA function-ships the START command to CICB.

5. CICB now processes the START command and, in doing so, tries to discover which region owns T1, because this is the region that has to execute the ATI request resulting from the START command.
6. CICB can determine where to send the ATI request only if a definition of T1, resulting from an earlier routed transaction, is present. Assuming that no such definition exists, the interval control program rejects the START request with TERMIDERR.

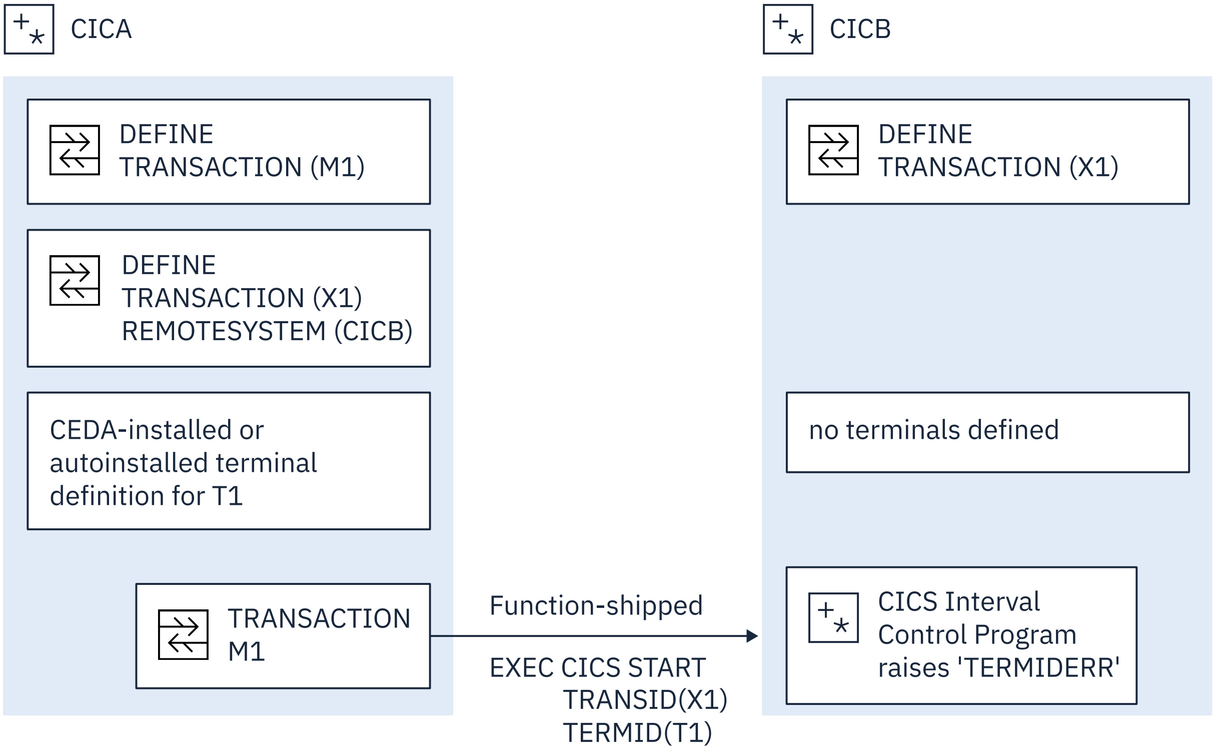

CICA
DEFINE TRANSACTION (M1)
DEFINE TRANSACTION (X1) REMOTESYSTEM (CICB)
CEDA-installed or autoinstalled terminal definition for T1
TRANSACTION M1
Function-shipped
EXEC CICS START TRANSID(X1) TERMID(T1)
CICB
DEFINE TRANSACTION (X1)
no terminals defined
CICS Interval Control Program raises 'TERMIDERR'

Figure 27. Failure of an ATI request in a system where the termid is unknown

The global user exits XICTENF and XALTENF

You, as user of the system, know how this routing problem could be solved, and CICS gives you a way of communicating your solution to the system. The two global user exits XICTENF and XALTENF have been provided.

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

The terminal allocation program schedules requests resulting both from the eventual execution of a START command and from the transient data queue trigger mechanism. This means that a START command could result in an invocation of both exits.

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

- Whether the ATI request resulted from: a START command with data, a START command without data, or a transient data queue trigger.
- Whether the START command was issued by a transaction that had been the subject of transaction routing.
- Whether the START command was function-shipped from another region.
- The identifier of the transaction to be run.
- The identifier of the terminal with which the transaction should run.
- The identifier of the terminal associated with the transaction that issued the START command, if this was a routed transaction, or the identifier of the session, if the command was function-shipped. Otherwise, blanks are returned.
- The netname of the last system the START request was shipped from or, if the START was issued locally, the netname of the system last transaction-routed from. Blanks are returned if no remote system was involved.
- The sysid corresponding to the returned netname.

On exit from the program, you tell CICS whether the terminal exists and, if it does, you supply either the netname or the sysid of the TOR. CICS sends the ATI request to the region you specify. As a result, the terminal definition is shipped from the TOR to the AOR, and transaction routing proceeds normally.

There is therefore a solution to the problem shown in Figure 27 on page 76. It is necessary only to write a small exit program that returns the CICS-supplied parameters unchanged and sets the return code for 'netname returned'.

The events that follow are shown in Figure 28 on page 78:

1. The interval control program accepts the START command and signals acceptance to the issuing system if this is required.
2. After the specified interval has expired, or immediately if no interval was specified, the terminal allocation program tries to schedule the ATI request. It finds no terminal defined and takes the exit XALTENF, which again supplies the required netname.
3. The ATI request is shipped to CICA. CICA allocates a relay transaction, establishes a transaction routing link to transaction X1 in CICB, and ships a copy of the terminal definition for T1 to CICB.

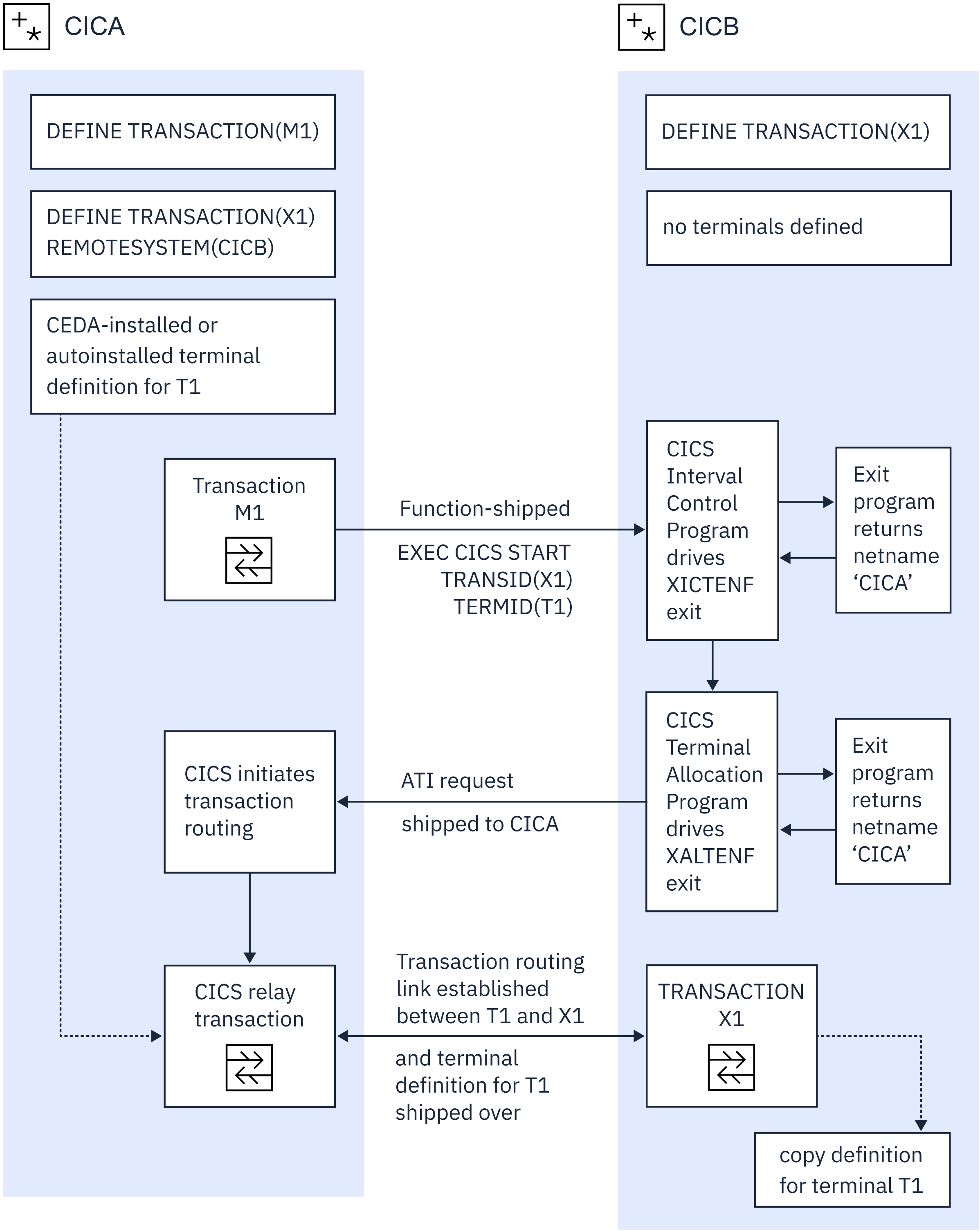

CICA
CICB
DEFINE TRANSACTION(M1)
DEFINE TRANSACTION(X1) REMOTESYSTEM(CICB)
CEDA-installed or autoinstalled terminal definition for T1
Transaction M1
Function-shipped
EXEC CICS START TRANSID(X1) TERMID(T1)
DEFINE TRANSACTION(X1)
no terminals defined
CICS Interval Control Program drives XICTENF exit
Exit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

Figure 28. Resolving a 'terminal not known' condition on a START request

The example in Figure 28 on page 78 shows only one of many possible configurations. From this elementary example, you can see how to approach a solution for the more complex situations that can arise in multiregion networks.

Resource definition
You do not have to be using autoinstalled terminals to make use of the exits XICTENF and XALTENF. The technique also works with terminals that you have defined explicitly, if they are defined with SHIPPABLE(YES) specified.

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

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

Important:

The XICTENF and XALTENF exits can be used only if there is a direct link between the AOR and the TOR. In other words, the sysid or netname that you pass back to CICS from the exit program must not be for an indirectly connected system.

The exit program for the XICTENF and XALTENF exits
How your exit program identifies the TOR from the parameters supplied by CICS can only be decided by reference to your system design.

In the simplest case, you would hand back to CICS the netname of the system that originated the START request. In a more complex situation, you may decide to give each terminal a name that reflects the system on which it resides.

For programming information about the exit program, see Terminal not known condition exits XALTENF and XICTENF. A sample program is also available in the DFHXTENF member of library CICSTS56.CICS.SDFHSAMP.

Shipping terminals for ATI from multiple TORs

Consider the following network setup:

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

Now consider the following scenario:

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

Figure 29 on page 80 illustrates what happens.

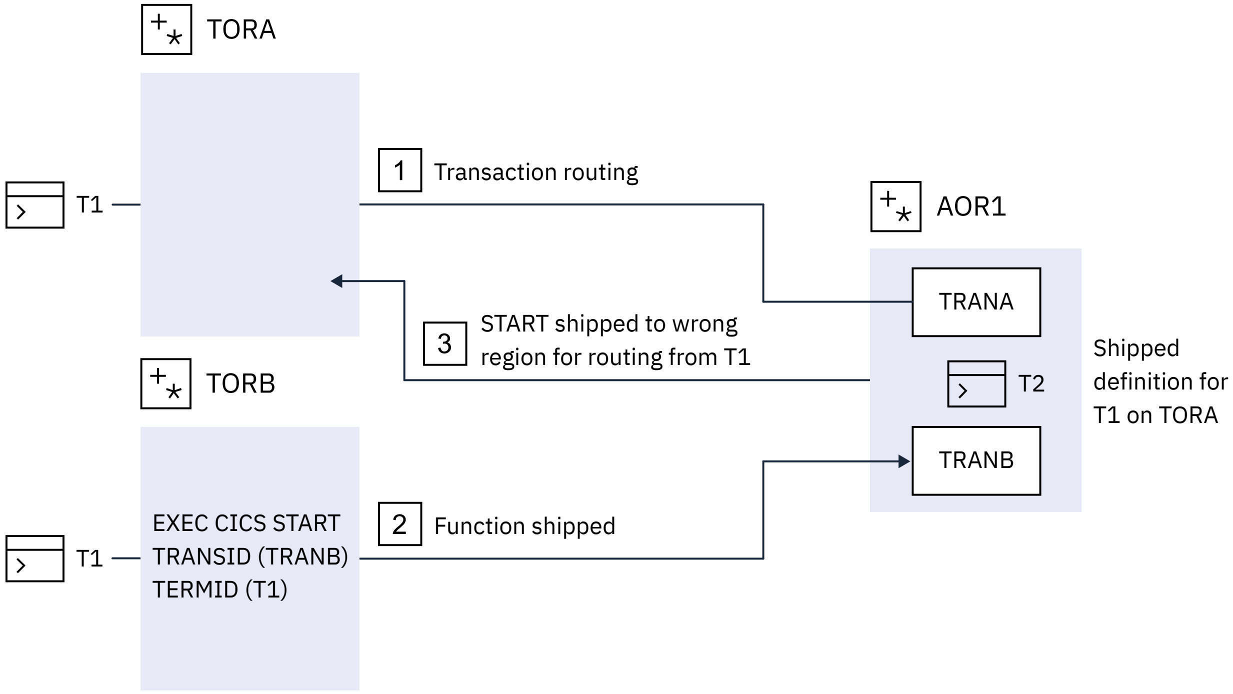

TORA
T1
1 Transaction routing
AOR1
TRANA
3 START shipped to wrong region for routing from T1
TORB
T2
Shipped definition for T1 on TORA
TRANB
EXEC CICS START TRANSID (TRANB) TERMID (T1)
2 Function shipped
T1

Figure 29. Function-shipped START request started against an incorrect terminal

There are two ways to prevent this situation:

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

   Use the enhanced routing method described in “Routing transactions invoked by START commands” on page 81. A terminal-related START command issued in the terminal-owning region is *not* function-shipped to the AOR; thus it cannot be shipped back to the wrong TOR. Instead, the START executes directly in the TOR, and the transaction is routed as if it had been initiated from a terminal.

   A definition of the terminal is shipped to the AOR, and the autoinstall user program is called. Your autoinstall user program can then allocate an *alias* termid in the AOR, to avoid a conflict with the previously installed remote definition. Terminal aliases are described in Terminal aliases. For information about writing an autoinstall program to control the installation of shipped definitions, see Writing a program to control autoinstall of shipped terminals.

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

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

   If the TOR to which the START request is returned is **not** the one referenced in the installed remote terminal definition, a definition of the terminal is shipped to the AOR, and the autoinstall user program is called. Your autoinstall user program can then allocate an alias termid in the AOR, to avoid a conflict with the previously installed remote definition.

   For full details of the **FSSTAFF** system initialization parameter, see FSSTAFF system initialization parameter.

ATI and generic resources

An AOR can issue an EXEC CICS START request against an LU that is owned by an SNA (z/OS Communications Server) generic resource, without knowing the member of the generic resource group to which the terminal is currently logged on.

For details of using ATI with generic resources, see Using ATI with generic resources.

Routing transactions invoked by START commands

To use the enhanced method of transaction routing for eligible terminal-related **EXEC CICS START** commands, the transaction must be defined as ROUTABLE(YES) in the requesting region (the region in which the **START** command is issued). The enhanced method of transaction routing supersedes the traditional method of transaction routing.

Note: You must use the traditional method for the following transactions:

- Transactions invoked by the trigger-level on a transient data queue
- Transactions that are invoked by **EXEC CICS START** commands but do not meet the requirements, as explained in “How to route transactions started by terminal-related START commands” on page 82, to use the enhanced method

Advantages of the enhanced method

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

Dynamic routing

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

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

Efficiency

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

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

Simplicity

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

Using the enhanced method, because a START command issued in a TOR is not function-shipped to the AOR, the “terminal-not-known” condition does not occur. The START command executes in the TOR directly, and the transaction is routed just as if it had been initiated from a terminal. If the terminal is not defined in the AOR, a definition is shipped from the TOR.

How to route transactions started by terminal-related START commands

You can set a number of options on a terminal-related START command that can affect the set of regions to which the transaction can be routed.

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

- The START command must be a member of the subset of eligible START commands; that is, it must meet all the following conditions:
  - The START command specifies the TERMID option, which names the terminal associated with the current task.
  - The principal facility of the task that issues the START command is a terminal. The principal facility is not a terminal if, for example, the program that issues the START command has a DPL link; in this case, the principal facility is the intersystem session.
  - The principal facility of the task that issues the START command is not a surrogate client virtual terminal.
  - The SYSID option of the START command does not specify the name of a remote region; that is, the remote region on which the transaction is to be started must not be specified explicitly.

  The requesting region and the TOR can be the same region.
- The requesting region and the TOR, if they are different, must be connected by one of the following links:
  - An MRO link
  - An APPC parallel-session link
  - An IPIC link. For IPIC links, both regions must be at CICS TS for z/OS, Version 4.1 or later
- The TOR and the target region must be connected by one of the following links:
  - An MRO link
  - An IPIC link

    For IPIC links, both regions must be at CICS TS for z/OS, Version 4.1 or later.
  - An APPC single-session or parallel-session link

    If an APPC link is used, at least one of the following must be true:

    Terminal-initiated transaction routing has previously taken place over the link.

CICSPlex SM is being used for routing.

- The transaction definition in the requesting region must specify ROUTABLE(YES).
- If the requesting region and the TOR are different, the transaction definition in the requesting region must not specify the REMOTESYSTEM option. If the requesting region and the TOR are the same region, you may use REMOTESYSTEM in the transaction definition for static routing.
- If the transaction is to be routed dynamically, the transaction definition in the TOR must specify DYNAMIC(YES).

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

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

START commands issued in an AOR

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

Static routing for commands issued in the AOR

Static routing takes place if the transaction definition in the application-owning region (AOR) specifies ROUTABLE(YES) and the transaction definition in the terminal-owning region (TOR) specifies DYNAMIC(NO). Therefore, the dynamic routing program is not called.

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

If the transaction is not eligible for enhanced routing, it is handled in the usual way, as described in "Traditional routing of transactions started by ATI" on page 72; that is, CICS tries to route it back to the originating AOR for execution. If the REMOTESYSTEM option of the transaction definition in the TOR names a region other than the originating AOR, the request fails.

Figure 30 on page 84 shows the requirements for using the enhanced method to statically route a transaction that is initiated by a terminal-related START command issued in an AOR.

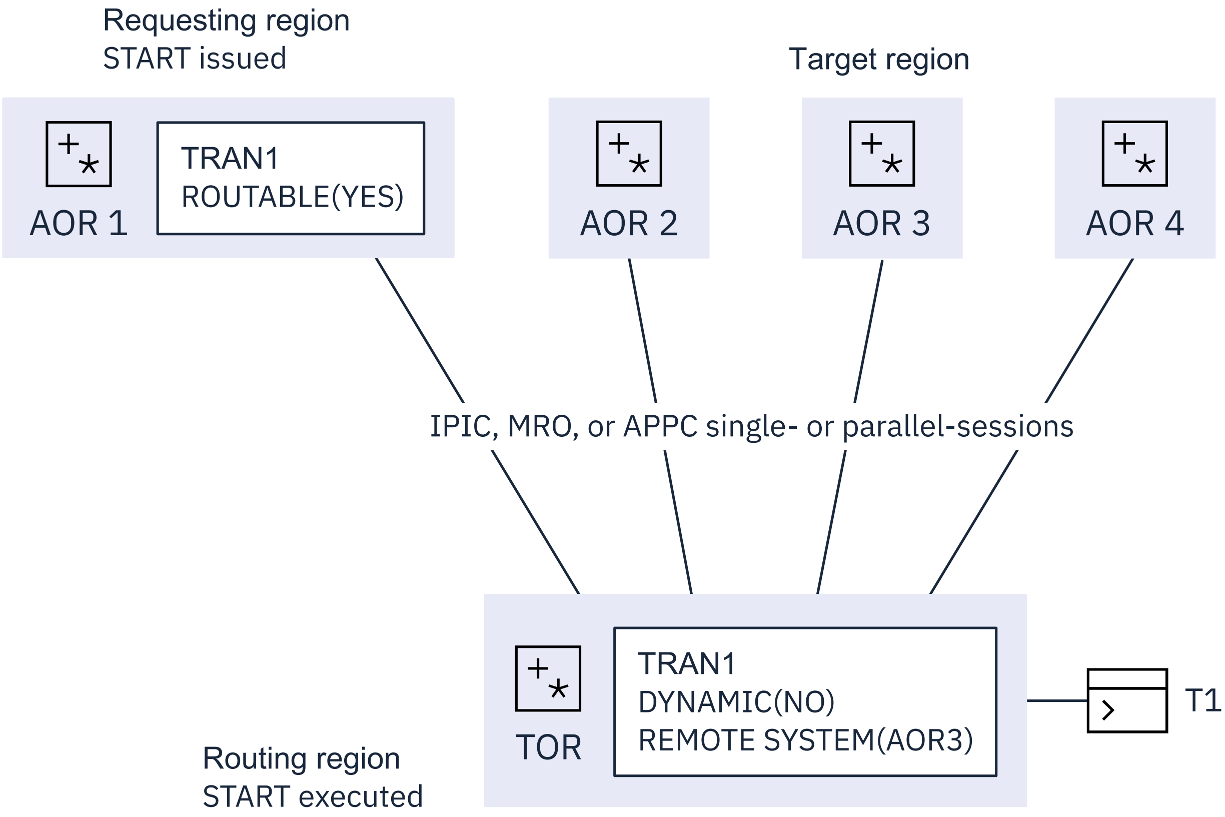


Figure 30. Static routing of a terminal-related START command issued in an AOR, using the enhanced method

The requesting region and the TOR are connected by an IPIC, MRO or APPC parallel-session link. The TOR and the target region are connected by an IPIC, MRO or APPC (single- or parallel-session) link. The transaction definition in the requesting region specifies ROUTABLE(YES). The transaction definition in the TOR specifies DYNAMIC(NO). The REMOTESYSTEM option names the AOR to which the transaction is to be routed.

Dynamic routing for commands issued in the AOR
Dynamic routing takes place if the transaction definition in the application-owning region (AOR) specifies ROUTABLE(YES) and the transaction definition in the terminal-owning region (TOR) specifies DYNAMIC(YES). Therefore, the dynamic routing program is invoked in the TOR.

Dynamic routing of transactions called by terminal-related START commands uses the *hub* routing model described in "The hub model" on page 62.

If the transaction is eligible for enhanced routing, the routing program can reroute the transaction to an alternative AOR; that is, to an AOR other than that in which the START was issued.

If the transaction is not eligible for enhanced routing, the dynamic routing program is called for notification only; it cannot reroute the transaction. The transaction is handled in the usual way; that is, it is routed back to the originating AOR for execution.

Figure 31 on page 86 shows the requirements for dynamically routing a transaction that is initiated by a terminal-related START command issued in an AOR.

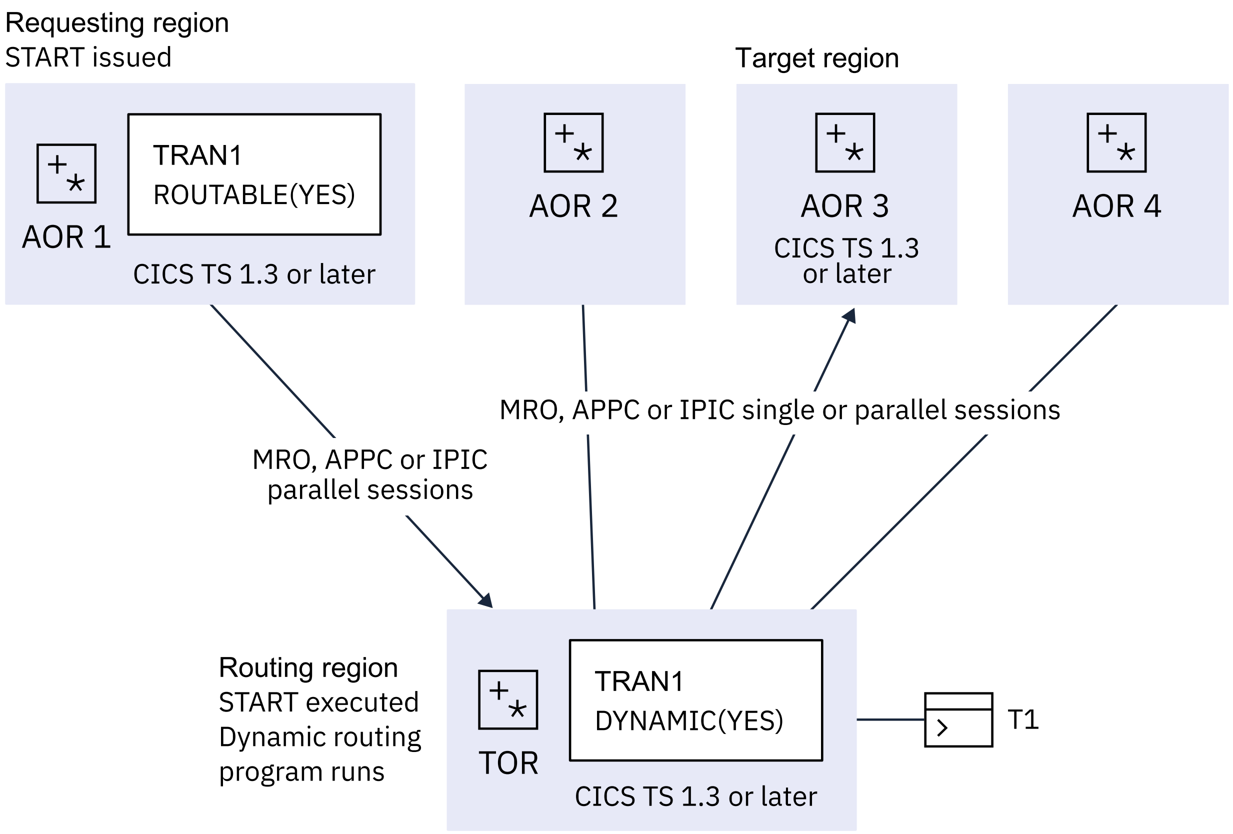


Figure 31. Dynamic routing of a terminal-related START command issued in an AOR

The requesting region and the TOR are connected by an MRO, APPC, or IPIC parallel-session link. The TOR and the target region are connected by an MRO, APPC, or IPIC (single-session or parallel-session) link. The transaction definition in the requesting region specifies ROUTABLE(YES). The transaction definition in the TOR specifies DYNAMIC(YES).

Dynamic routing using IPIC is supported in CICS TS 4.1 or later.

START commands issued in a TOR

A terminal-related START command that is issued in a TOR can be statically or dynamically routed.

Static routing of terminal-related START commands

To statically route transactions using the enhanced method, specify ROUTABLE(YES) and DYNAMIC(NO) in the transaction definition in the terminal-owning region, so that the dynamic routing program is not called.

If the transaction is eligible for enhanced routing, the following steps take place:

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

If the transaction is not eligible for enhanced routing, the START request is handled in the usual way, described in "Traditional routing of transactions started by ATI" on page 72; that is, it is function-shipped to the AOR named in the REMOTESYSTEM option of the transaction definition. If REMOTESYSTEM is not specified, the START request runs locally in the TOR.

Figure 32 on page 88 shows the requirements for using the enhanced method to statically route a transaction that is initiated by a terminal-related START command issued in a TOR.

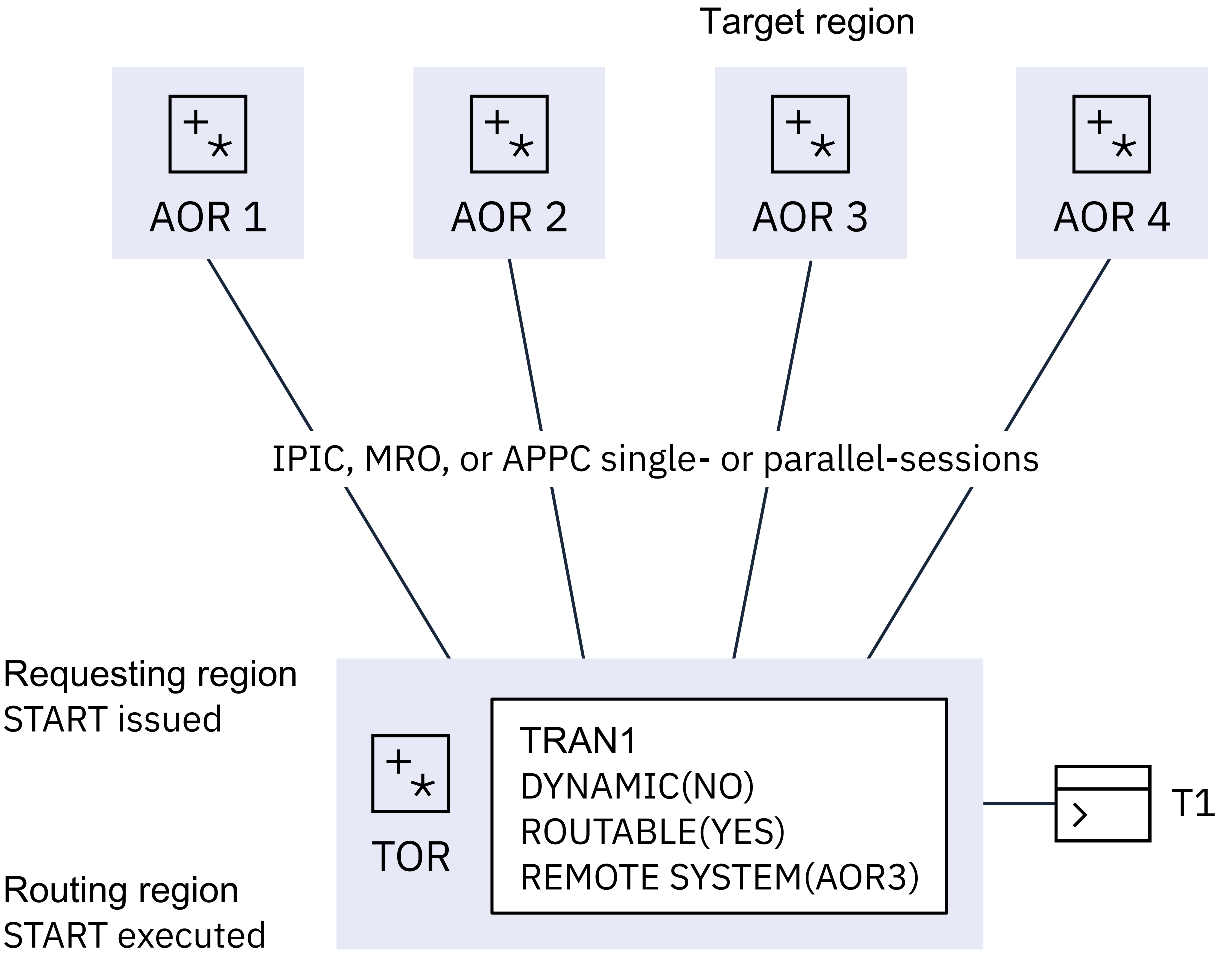


Figure 32. Static routing of a terminal-related START command issued in a TOR, using the enhanced method

The TOR and the target region are connected by an IPIC, MRO or APPC (single or parallel session) link. The transaction definition in the TOR specifies DYNAMIC(NO) and ROUTABLE(YES). The REMOTESYSTEM option names the AOR to which the transaction is to be routed.

Dynamic routing of terminal-related START commands

To dynamically route transactions using the enhanced method, specify ROUTABLE(YES) and DYNAMIC(YES) in the transaction definition in the terminal-owning region, so that the dynamic routing program is called.

Dynamic routing of transactions started by terminal-related START commands use the hub routing model.

If the transaction is eligible for enhanced routing, the following steps take place:

1. The START command runs in the TOR.
2. The routing program can route the transaction.

If the transaction is not eligible for enhanced routing, the dynamic routing program is called for notification only, because it cannot route the transaction. The START request is handled in the usual way; that is, it is function-shipped to the AOR named in the REMOTESYSTEM option of the transaction definition in the TOR. If REMOTESYSTEM is not specified, the START request runs locally in the TOR.

Figure 33 on page 90 shows the requirements for dynamically routing a transaction that is initiated by a terminal-related START command issued in a TOR.

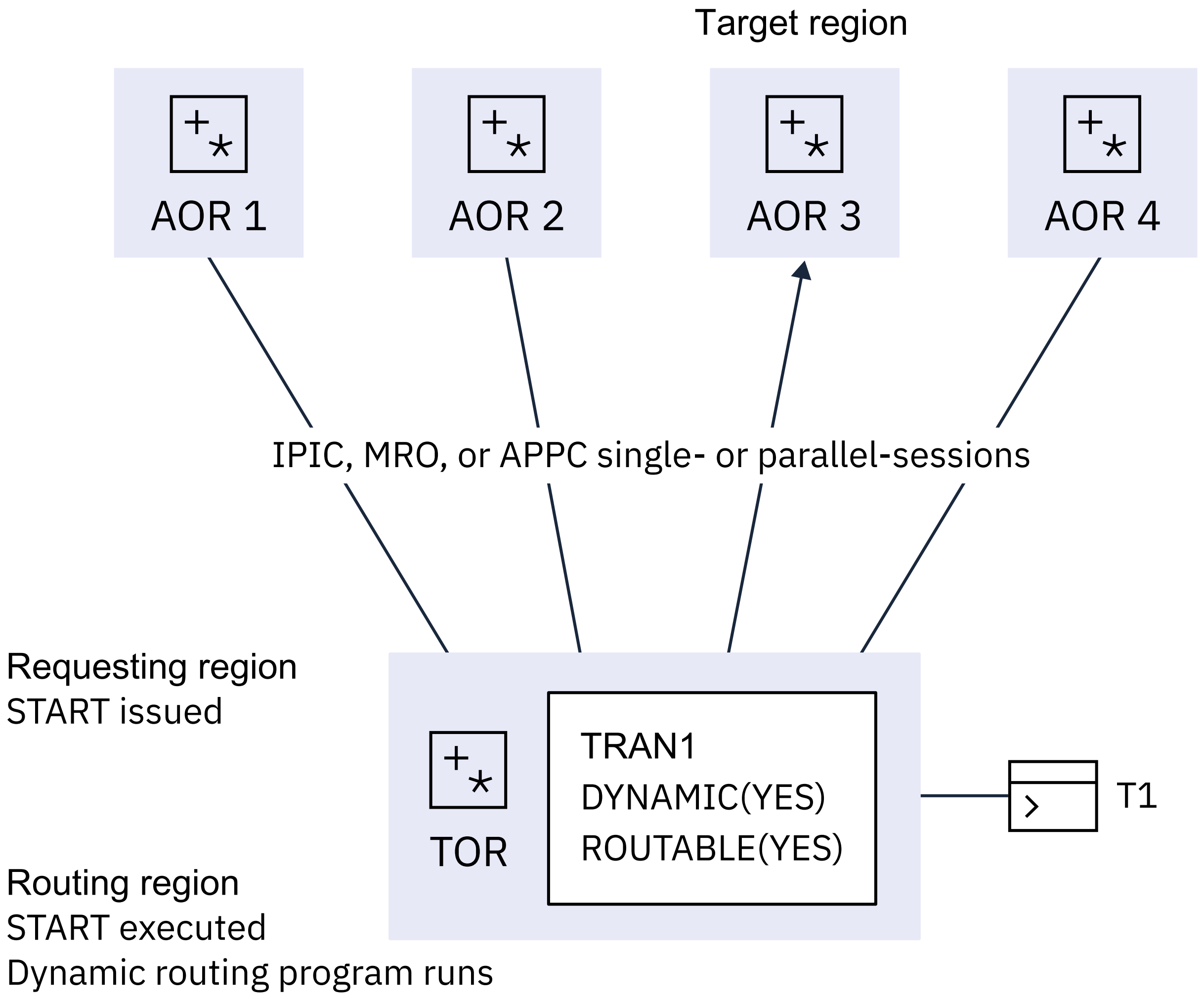


Figure 33. Dynamic routing of a terminal-related START command issued in a TOR

The TOR and the target region are connected by an IPIC, MRO, or APPC (single or parallel session) link. The transaction definition in the TOR specifies both DYNAMIC(YES) and ROUTABLE(YES).

How to route non-terminal-related START commands

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

- The requesting region and the target region are connected in one of the following ways:
  - An MRO link.
  - An APPC single-session or parallel-session link. If an APPC link is used and the distributed routing program is called on the target region, CICSPlex SM must be used for routing.
  - An IPIC link.
- The transaction definition in the requesting region specifies DYNAMIC(YES).

In addition, if the request is to be routed dynamically, the SYSID option of the START command must not specify the name of a remote region. (That is, the remote region on which the transaction is to be started must not be specified explicitly.)

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

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

  If the request is delayed, the interval control element (ICE) created by the START request is kept in the requesting region with a transaction ID of CDFS. When the ICE expires, the CDFS transaction runs; it retrieves any data specified by the user and reissues the START request without an interval. The request will be routed based on the state of the transaction definition and the sysplex at that moment. Ensure that security for CDFS is correctly configured.
- QUEUE.
- REQID.
- RTERMID.
- RTRANID.

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

Static routing of non-terminal-related START requests

The transaction definition in the requesting region specifies DYNAMIC(NO), which means that the distributed routing program is not invoked. The transaction is statically routed to the region that is specified in the REMOTSYSTEM option of the transaction definition. If REMOTESYSTEM is not specified, the transaction runs locally.

Dynamic routing of non-terminal-related START requests

Dynamic routing of non-terminal-related START requests uses the distributed routing model. The transaction definition in the requesting region specifies DYNAMIC(YES). If the request is eligible for dynamic routing, the distributed routing program is invoked for routing. The START request is function-shipped to the target region returned by the routing program.

Note:

1. If the request is ineligible for dynamic routing, the distributed routing program is not invoked. Unless the SYSID option specifies a remote region explicitly, the START request is function-shipped to the AOR named in the REMOTESYSTEM option of the transaction definition in the requesting region; if REMOTESYSTEM is not specified, the START executes locally, in the requesting region.
2. If the request is eligible for dynamic routing, but the SYSID option of the START command names a remote region, the distributed routing program is invoked for notification only; it cannot route the request. The START executes on the remote region named on the SYSID option.

3. If the return code from the distributed routing program is not zero, the request fails with the following response code:

   ```
   eibrcode=SYSIDERR, eibresp2=1
   ```

   The dynamic routing program rejected the START request.

Canceling interval control requests
To cancel a previously-issued START, DELAY, or POST interval control request, you use the CANCEL command.

About this task

The REQID option specifies the identifier of the request to be canceled. If the request is due to execute on a remote region, you can use the SYSID option to specify that the CANCEL command is to be shipped to that region.

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

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

   a. Checks:

      i) That it has been invoked for a CANCEL command.

      ii) That the SYSID option was not specified on the command.

      iii) That the identifier of the request to be canceled does not begin with 'DF'. ('DF' indicates a request issued internally by CICS.)

      iv) That the name of the transaction that issued the CANCEL command does not begin with 'C'—that is, that the transaction is not a CICS internal transaction, nor a CICS-supplied transaction such as CECI.

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

   b. Instructs CICSPlex SM to:

      i) Search every CICS region that it knows about for an interval control request with the identifier (REQID) specified on the CANCEL command.

      ii) On each region, cancel the first request (with the specified identifier) that it finds. Note that:

         - Requests may be canceled on more than one region.
         - If a particular region contains more than one request with the specified identifier, only the first request found by CICSPlex SM is canceled.
         - You must ensure that CICSPlex SM has UPDATE access to the transaction ID of the transaction associated with the CANCEL request.

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

For details of the CANCEL command, see CANCEL. For general information about how to write an XICEREQ global user exit program, see Interval control EXEC interface program exits (XICEREQ, XICERES, and XICEREQC).

Allocation of remote APPC connections

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

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

Transaction routing with APPC devices

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

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

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

You use the APPC application program interface to communicate with APPC devices. For relevant introductory information, see “Distributed transaction processing” on page 109.

Allocating an alternate facility

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

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

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

The system as a terminal

Because the resource definitions for APPC devices can take the CONNECTION and SESSIONS form, it is easy to confuse them with the definitions for the intersystem links.

It is important to remember that definitions for the intersystem links are either **direct** or **indirect**, while those for APPC devices are **direct** in the TOR and **remote** in the AOR and any intermediate systems. Note also that remote CONNECTION definitions do not need corresponding SESSIONS definitions.

Figure 34 on page 94 shows a network of three CICS systems chained together, of which the first is linked to an APPC terminal.

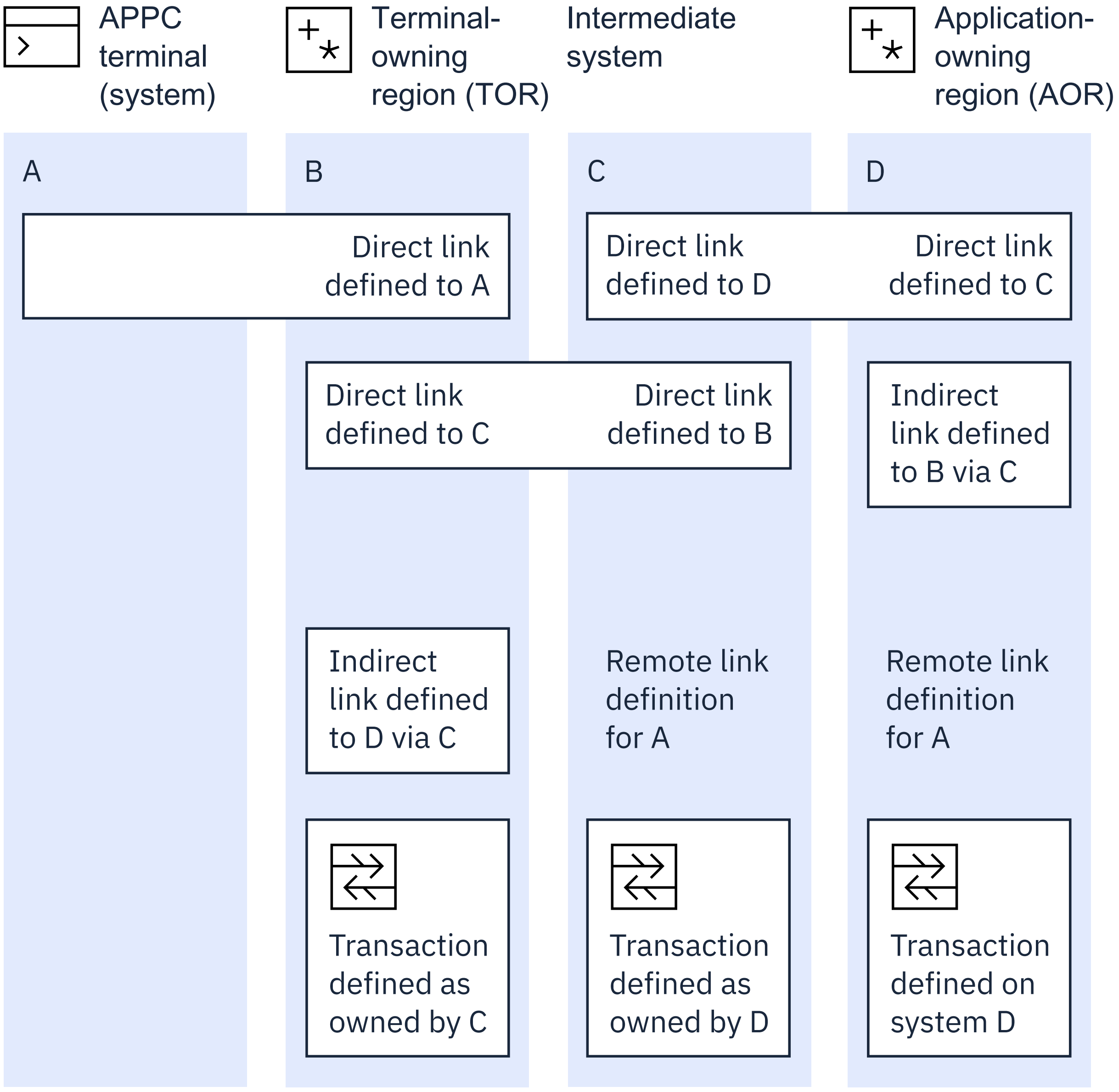

APPC terminal (system)
Terminal-owning region (TOR)
Intermediate system
Application-owning region (AOR)
A
B
C
D
Direct link defined to A
Direct link defined to D
Direct link defined to C
Direct link defined to C
Direct link defined to B
Indirect link defined to B via C
Indirect link defined to D via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

Figure 34. Transaction routing to an APPC terminal across daisy-chained systems

Note:

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

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

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

Note these points:

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

The relay program

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

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

Because it executes the relay program, the transaction is called the **relay transaction**.

When the relay transaction is attached, it acquires an interregion or intersystem session and sends a request to the remote system to cause the “real” user transaction to be started. In the application-owning region, the terminal is represented by a control block known as the **surrogate** TCTTE. This TCTTE becomes the transaction's principal facility, and is indistinguishable by the transaction from a “real” terminal entry. However, if the transaction issues a request to its principal facility, the request is intercepted by the CICS terminal control program and shipped back to the relay transaction over the interregion or intersystem session. The relay transaction then issues the request or output to the terminal. In a similar way, terminal status and input are shipped through the relay transaction to the user transaction.

Automatic transaction initiation (ATI) is handled in a similar way. If a transaction that is initiated by ATI requires a terminal that is connected to another system, a request to start the relay transaction is sent to the terminal-owning region. When the terminal is free, the relay transaction is connected to it.

The relay transaction remains in existence for the life of the user transaction and has exclusive use of the session to the remote system during this period. When the user's transaction terminates, an indication is sent to the relay transaction, which then also terminates and frees the terminal.

Basic mapping support (BMS)

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

For BMS page building and routing requests, the pages are built and stored in the AOR. When the logical message is complete, the pages are shipped to the terminal-owning region (or regions, if they were generated by a routing request), and deleted from the AOR. Page retrieval requests are processed by a BMS program running in the system to which the terminal is connected.

BMS message routing to remote terminals and operators

You can use the BMS ROUTE command to route messages to remote terminals.

For programming information about the BMS ROUTE command, see ROUTE. You cannot, however, route a message to a selected remote operator or operator class unless you also specify the terminal at which the message is to be delivered.

In all cases, the remote terminal must be defined in the system that issues the ROUTE command (or a shipped terminal definition must already be available; see Shipping terminal and connection definitions). Note that the facility described in “Shipping terminals for automatic transaction initiation” on page 74 does not apply to terminals addressed by the ROUTE command.

Table 2. BMS message routing to remote terminals and operators

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

Using the routing transaction, CRTE

The routing transaction, CRTE, is a CICS-supplied transaction used by a terminal operator to call transactions that are owned by a connected CICS system. CRTE facility is particularly useful for testing remote transactions before final installation.

CRTE can be used from any 3270 display device.

To use CRTE, the terminal operator enters:

```
CRTE SYSID=xxxx  [TRPROF={DFHCICSS|profile_name}]
```

where:

- *xxxx* is the name of the CONNECTION or the first four characters of the IPCONN resource that defines the connection to the remote system
- *profile_name* is the name of the profile to be used for the session with the remote system

See Defining communication profiles for more information about defining profiles. The transaction then indicates that a routing session has been established, and the user enters input of the form:

```
yyyyzzzzzz...
```

where *yyyy* is the name by which the required remote transaction is known on the remote system, and *zzzzzz...* is the initial input to that transaction. Subsequently, the remote transaction can be used as if it had been defined locally and called in the ordinary way. All further input is directed to the remote system until the operator terminates the routing session by entering CANCEL.

In secure systems, operators are typically required to sign on before they can start transactions. The first transaction that is called in a routing session is therefore usually the sign-on transaction CESN; that is, the operator signs on to the remote system.

Although the routing transaction is implemented as a pseudoconversational transaction, the terminal from which it is called is held by CICS until the routing session ends. Any ATI requests that name the terminal are therefore queued until the CANCEL command is issued.

System programming for transaction routing

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

Procedure

1. Install MRO or ISC support, or both.
2. Define MRO or ISC links between the systems that are to be connected, as described in Defining connections to remote systems.
3. Define the terminals and transactions that will participate in transaction routing, as described in Defining remote resources.
4. Ensure that the local communication profiles, transactions, and programs required for transaction routing are defined and installed on the local system, as described in Defining remote resources.
5. If you want to use dynamic transaction routing, customize the supplied dynamic routing program, DFHDYP, or write your own version.

   For programming information about how to do this, see Writing a dynamic routing program.
6. If you want to route to shippable terminals from regions where those terminals might be 'not known', code and enable the global user exits XICTENF and XALTENF.

   For programming information about coding these exits, see Terminal not known condition exits XALTENF and XICTENF.

What to do next

Control intersystem queuing

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

For guidance information about controlling intersystem queues, see Intersystem session queue management.

CICS distributed program link

This chapter describes CICS distributed program link (DPL).

It contains:

- "Overview of DPL" on page 98
- "Statically routing DPL requests" on page 100
- "Dynamically routing DPL requests" on page 103
- "Limitations of DPL server programs" on page 107
- "Intersystem queuing" on page 107
- "Examples of DPL" on page 108.

Overview of DPL

CICS distributed program link enables CICS application programs to run programs that are in other CICS regions by shipping program-control LINK requests.

An advantage of DPL is that you can write an application without knowledge of the location of the requested programs. The application uses program-control **LINK** commands in the usual way. The CICS program resource definitions usually specify that the named program is not in the local region (*client region*), but in a remote region (*server region*).

An illustration of a DPL request is shown in Figure 35 on page 99. In this diagram, a program (the *client program*) running in CICA issues a program-control LINK command for a program called PGA (the *server program*). From the installed program definitions, CICS discovers that the PGA program is owned by a remote CICS system called CICB. CICS changes the LINK request into a suitable transmission format and then ships it to CICB to run.

In CICB, the mirror transaction (described in "The mirror transaction and transformer program" on page 43) is attached. The mirror program DFHMIRS, which is used by all mirror transactions, re-creates the original request and issues the request on CICB. When the server program has run to completion, the mirror program returns any communication-area data to CICA.

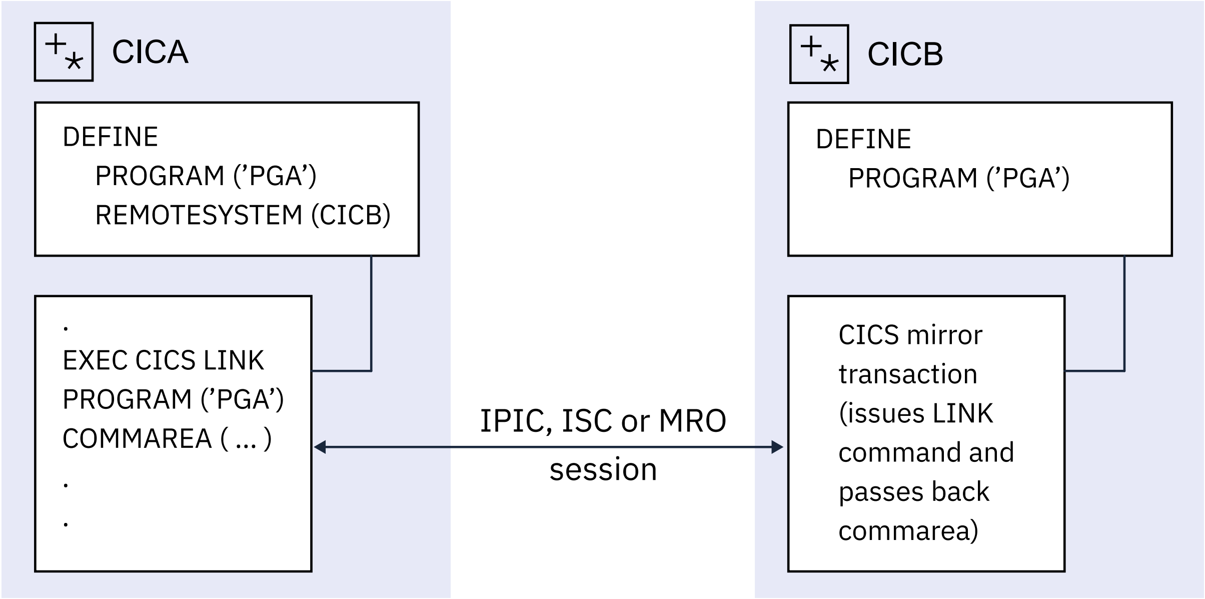


Figure 35. Distributed program link

The CICS recovery and restart facilities enable resources in remote regions to be updated and ensure that, when the client program reaches a sync point, any mirror transactions that are updating protected resources also take a sync point. So changes to protected resources in remote and local systems are consistent. The CSMT transient data queue is notified of any failures in this process, so that suitable corrective action can be taken, whether manually or by user-written code.

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

For regions from CICS TS for z/OS, Version 4.2, when you use an IPIC connection and a long-running mirror, CICS runs the mirror program DFHMIRS on an L8 open TCB whenever possible, which can improve performance for threadsafe programs in the server region. The LINK command also is threadsafe when it is used to link to a program in a remote CICS region over an IPIC connection only. For MRO and ISC over SNA connections, the mirror program does not run on an open TCB and the LINK command is not threadsafe.

If both an IPIC connection and an ISC over SNA connection exist between two CICS regions, and both have the same name, the IPIC connection takes precedence. That is, if remote region CICB is defined by both an IPCONN definition and a CONNECTION definition, CICS uses the IPCONN definition. However, if the IPCONN is not acquired but is in service, the ISC over SNA connection is used.

Statically routing DPL requests

Static routing means that the location of the server program is specified at design time, rather than at run time. DPL requests for a particular remote program are always routed to the same server region. Typically, when static routing is used, the location of the server program is specified in the PROGRAM resource.

The program resource definition can also specify the name of the server program as it is known on the resource system, if it is different from the name by which it is known locally. When the server program is requested by its local name, CICS substitutes the remote name before sending the request. This facility is useful when a server program exists with the same name on more than one system, but performs different functions depending on the system on which it is located.

Consider, for example, a local system CICA and two remote systems CICB and CICC. A program named PG1 resides in both CICB and CICC. These two programs are defined in CICA, but, because they have the same name, a local alias and a REMOTENAME must be defined for at least one of the programs. For example:

- Definition of program PG1 in system CICB:

  PROGRAM(PG1)
  REMOTESYSTEM(CICB)

- Definition of program PG1 in system CICC, that uses a local alias of PG99 and the REMOTENAME attribute:

  PROGRAM(PG99)
  REMOTENAME(PG1)
  REMOTESYSTEM(CICC)

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

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

In the client region (CICA in Figure 36 on page 102), the command-level EXEC interface program determines that the requested server program is on another system (CICB in the example). It therefore calls the transformer program to transform the request into a form suitable for transmission (in the example, line (2) indicates this). As indicated by line (3) in the example, the EXEC interface program then calls on the intercommunication component to send the transformed request to the appropriate connected system.

Using the mirror transaction

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

If you use a user-specified name for the mirror transaction initiated by any given DPL request, the following actions become easier:

- Controlling access to resources
- Accounting for system usage
- Performance tuning
- Establishing an audit trail

This transaction name must be defined in the server region as a transaction that invokes the mirror program DFHMIRS. If you define user transactions to invoke the mirror program, you can then specify appropriate values for all the other options on the transaction resource definition. To initiate any user-defined mirror transaction, the client program specifies the transaction name on the LINK request. Alternatively, the transaction name can be specified on the TRANSID option of the program resource definition.

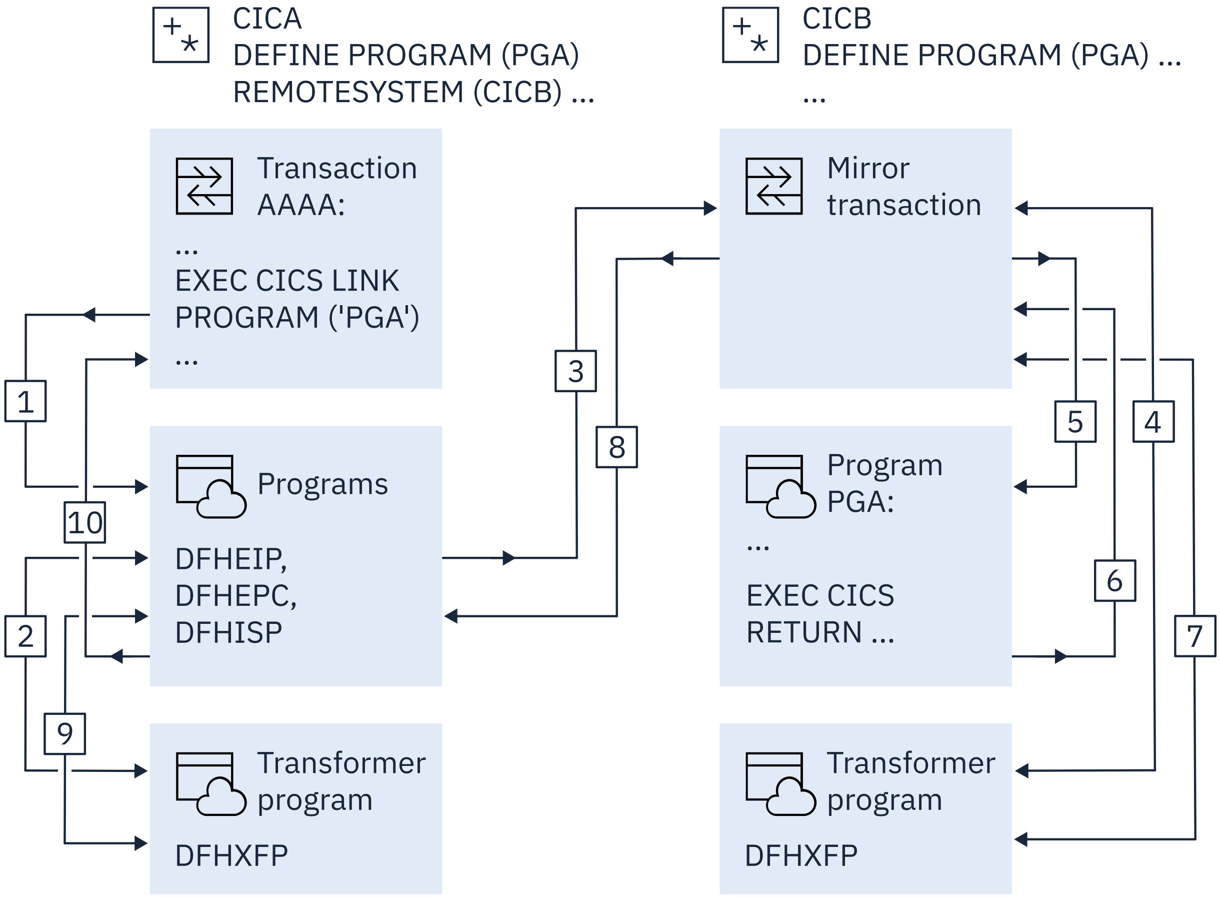


Figure 36. The transformer program and the mirror in DPL

As line (4) in Figure 36 on page 102 shows, a mirror transaction uses the transformer program DFHXFP to decode the formatted link request. The mirror then executes the corresponding command, thereby linking to the server program PGA (5). When the server program issues the RETURN command (6), the mirror transaction uses the transformer program to construct a formatted reply (7). The mirror transaction returns this formatted reply to the client region (8). In that region (CICA in the example), the reply is decoded, again using the transformer program (9), and used to complete the original request made by the client program (10).

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

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

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

Using global user exits to redirect DPL requests

Two global user exits can be invoked during DPL processing.

About this task

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

XPCREQ and XPCREQC can be used for a variety of purposes. You could, for example, use them to route DPL requests to different CICS regions, thereby providing a simple load balancing mechanism. However, a better way of doing this is to use the CICS dynamic routing program—see "Dynamically routing DPL requests" on page 103.

For programming information about writing XPCREQ and XPCREQC global user exit programs, see Program control program exits (XPCREQ, XPCERES, XPCREQC, XPCFTCH, XPCHAIR, XPCTA, and XPCABND).

Dynamically routing DPL requests

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

Dynamic routing models:

Dynamic routing of DPL requests received from outside CICS uses the "hub" routing model described in "The hub model" on page 62.

Dynamic routing of CICS-to-CICS DPL requests uses the distributed routing model described in "The distributed model" on page 64. Note, however, that it is the *dynamic* routing program, not the distributed routing program, that is invoked for routing CICS-to-CICS DPL requests.

For eligible DPL requests, a user-replaceable program called the *dynamic routing program* is invoked. (This is the same dynamic routing program that is invoked for transactions defined as DYNAMIC—see "Dynamic transaction routing" on page 70.) The routing program selects the server region to which the program-link request is shipped.

The default dynamic routing program, supplied with CICS, is named DFHDYP. You can modify the supplied program, or replace it with one that you write yourself. You can also use the DTRPGM system initialization parameter to specify the name of the program that is invoked for dynamic routing, if you want to name your program something other than DFHDYP. For programming information about user-replaceable programs in general, and about the dynamic routing program in particular, see Writing a dynamic routing program.

If you are using a threadsafe program that makes DPL requests that are transmitted to another region using IPIC communication, you might benefit from improved performance by changing your dynamic routing program to be coded to threadsafe standards.

You can review the value of the CONCURRENCY attribute in the PROGRAM resource definition for your dynamic routing program. If the program is not defined as threadsafe, each use of the program causes a switch back to the QR TCB, incurring an additional cost. If the program is defined as threadsafe but uses non-threadsafe CICS commands (which is permitted), each non-threadsafe command causes a switch back to the QR TCB and incurs the additional cost. For more information about threadsafe programs, see Threadsafe programs.

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

Which requests can be dynamically routed?

For a program-link request to be eligible for dynamic routing, the remote program must either be defined to the local system as DYNAMIC(YES) or not be defined to the local system.

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

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

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

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

- Calls from the CICS Web Interface
- Calls from CICS Transaction Gateway
- Calls from external CICS interface (EXCI) client programs
- ONC/RPC calls
- API/Service calls from z/OS Connect Enterprise Edition
- Calls from the CICSRequest node in IBM Integration Bus or IBM App Connect Enterprise

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

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

When the dynamic routing program is invoked

Program-link requests are both “traditional” CICS-to-CICS DPL calls and requests received from outside CICS. For eligible program-link requests the dynamic routing program is invoked at the following points.

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

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

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

Using CICSPlex SM to route requests

If you use CICSPlex SM to manage your CICSplex, you might not need to write your own dynamic routing program. CICSPlex SM provides a dynamic routing program that supports both workload routing and workload separation. All you have to do is to tell CICSPlex SM which regions in the CICSplex can participate in dynamic routing.

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

How CICS obtains the transaction ID

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

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

If you write your own dynamic routing program, perhaps based on DFHDYP, the transaction ID associated with the request might not be significant; you could, for example, code your program to route requests based on program name and available AORs (application owning regions).

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

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

Daisy-chaining of DPL requests

Daisy-chaining is supported for statically-routed DPL requests, but not supposed to be used for dynamically-routed DPL requests.

Support for statically-routed DPL requests

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

Considerations for dynamically-routed DPL requests

Dynamically-routed DPL requests are not supposed to be daisy-chained from region to region, and attempts to do so might cause unpredictable results.

Imagine two CICS regions, A and B. A program P is defined as DYNAMIC(YES) in both regions. In region B, P is defined with the attribute REMOTESYSTEM set to either the local SYSID or blank but not to any other value [Note]. An **`EXEC CICS LINK PROGRAM(P)`** command is issued in region A. The dynamic routing program is invoked in region A and routes the request to region B. In region B, the dynamic routing program is not invoked, even though program P is defined as DYNAMIC(YES); CICS attempts to run P locally in region B.

CICS does not support the daisy-chaining of dynamic DPL requests which includes combining dynamic routing with static routing. Attempts to do so might cause unpredictable results. When a DPL request has been dynamically routed, CICS expects the program to execute in the target region and not be routed somewhere else. If CICSPlex SM is used as the routing program, CICSPlex SM uses all the system data available to the routing region for it to select a suitable target; in this case, it does not make sense for the request to be routed again elsewhere.

In summary, in the case of dynamically-routed DPL requests, any further routing, whether dynamic or static, after a DPL is not supported by CICS.

Note: If in region B program P is defined with REMOTESYSTEM set to a remote region, CICS will be forced to honor this attribute and statically route the program to the specified region. However, this is not supported by CICS and might cause unpredictable results.

Routing program-link requests from outside CICS

Traditional CICS-to-CICS distributed program link (DPL) calls, instigated by **`EXEC CICS LINK PROGRAM`** commands, can be *daisy-chained* from a CICS region to another CICS region by defining the program as remote in each region except the last (server) region, where it is to execute. The same applies to program-link requests received from outside CICS.

For example, all of the following types of program-link requests from outside CICS can be routed:

- Calls received from:
  - CICS Web support
  - The CICS Transaction Gateway
- Calls from external CICS interface (EXCI) client programs
- External Call Interface (ECI) calls from any of the CICS Client workstation products
- ONC/RPC calls.

Using static routing for program-link requests from outside CICS

A program-link request received from outside CICS can be statically routed to a remote CICS region by specifying the name of the remote region on the REMOTESYSTEM option of the installed program definition.

Using dynamic routing for program-link requests from outside CICS

A program-link request received from outside CICS can be dynamically routed by defining the program to CICS as DYNAMIC(YES), and coding your dynamic routing program to route the request.

Limitations of DPL server programs

A DPL server program cannot issue the following types of commands.

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

If the client specifies SYNCONRETURN:

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

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

For further information about application programming for DPL, see Application programming for CICS DPL.

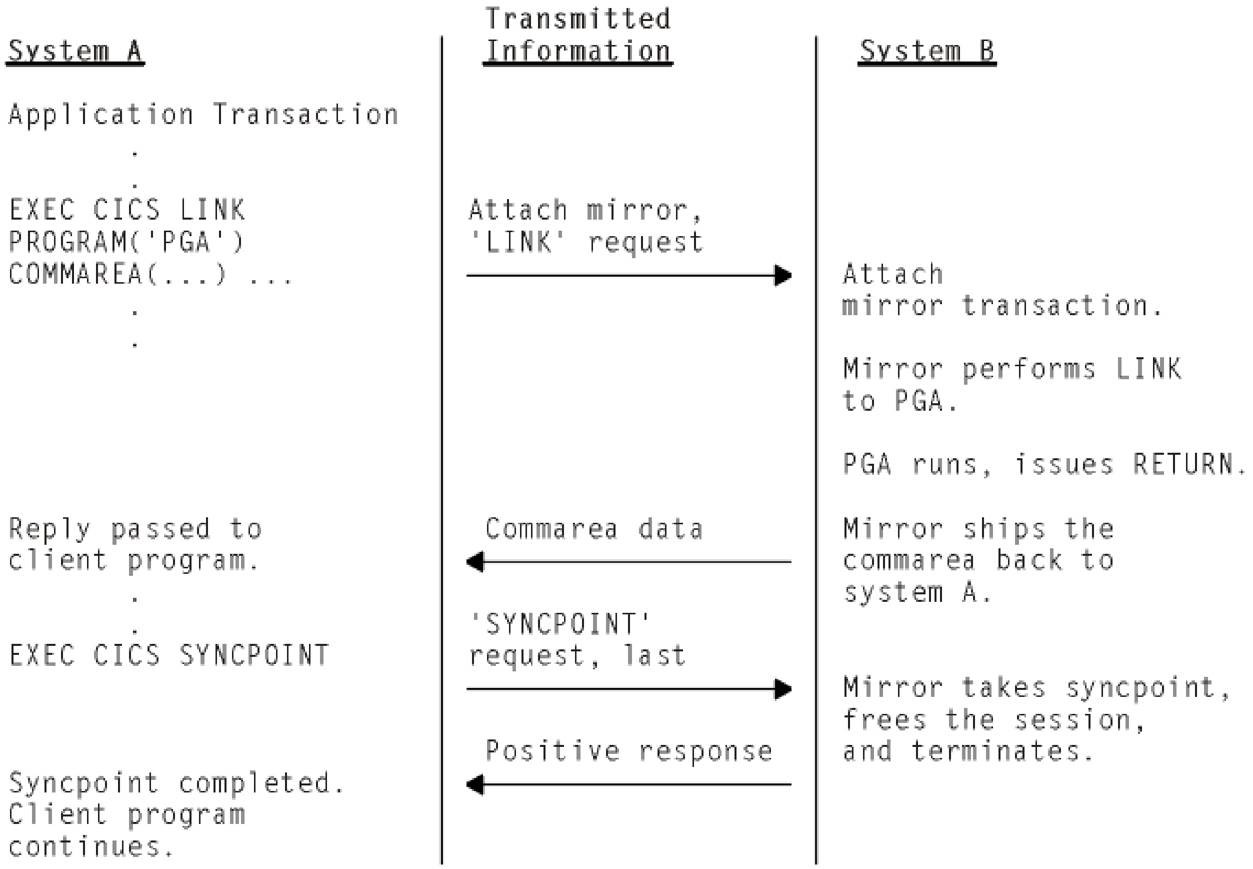
Intersystem queuing

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

For guidance information about controlling intersystem queues, see Intersystem session queue management.

Examples of DPL

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



```
                             Transmitted
System A                     Information          System B

Application Transaction
       .
       .
EXEC CICS LINK               Attach mirror,
PROGRAM('PGA')               'LINK' request
COMMAREA(...) ...            ------------->       Attach
       .                                          mirror transaction.
       .
                                                  Mirror performs LINK
                                                  to PGA.

                                                  PGA runs, issues RETURN.

Reply passed to               Commarea data       Mirror ships the
client program.              <-------------       commarea back to
       .                                          system A.
       .                     'SYNCPOINT'
EXEC CICS SYNCPOINT          request, last
                             ------------->       Mirror takes syncpoint,
                                                  frees the session,
                              Positive response   and terminates.
Syncpoint completed.         <-------------
Client program
continues.
```

Figure 37. DPL with the client transaction issuing a syncpoint

Figure 37 on page 108 shows a DPL request on which the client transaction issues a syncpoint. Because the mirror is always long-running, it does not terminate before SYNCPOINT is received.

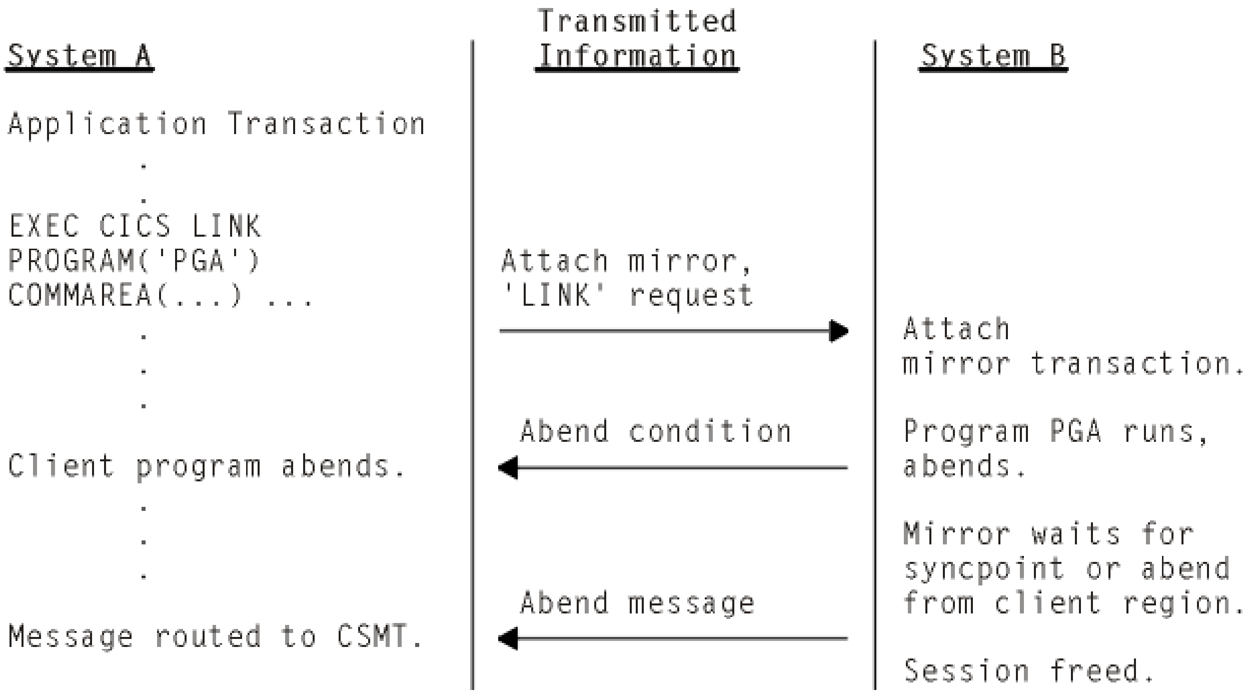


Figure 38. DPL with the server program abending

Figure 38 on page 109 shows a DPL request on which the server program abends.

Distributed transaction processing

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

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

Advantages over function shipping and transaction routing

Distributed transaction processing has advantages over function shipping and transaction routing.

Function shipping gives access to remote resources, and transaction routing lets a terminal communicate with remote transactions. These two facilities might appear sufficient for all your intercommunication needs, especially from a functional perspective. However, you must consider other design criteria, for example, machine loading, response time, continuity of service, and economic use of resources.

Consider the example of a supermarket chain. It has many branches that each stock a different range of goods, which are served by several distribution centers. Local stock records at the branches are updated online from point-of-sale terminals. Sales information must be sorted for the separate distribution centers, and transmitted to them to enable reordering and distribution.

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

- Data is transmitted to the remote systems irregularly in small packets. This means inefficient use of the links.

- The transactions associated with the point-of-sale devices are competing for sessions with the remote systems. This could mean unacceptable delays at point-of-sale.
- Failure of a link results in a catastrophic suspension of operations at a branch.
- Intensive intercommunication activity (for example, at peak periods) causes reduction in performance at the terminals.

Now consider the solution where each sales transaction writes its reorder records to a transient data queue. The data is quickly disposed of, leaving the transaction to carry on its conversation with the terminal.

Restocking requests are seldom urgent, so it might be possible to delay the sorting and sending of the data until an off-peak period. Alternatively, the transient data queue could be set to trigger the sender transaction when a predefined data level is reached. Either way, the sender transaction has the same job to do.

Again, it is tempting to use function shipping to transmit the reorder records. After the sort process, each record could be written to a remote file in the relevant remote system. However, this method is still not ideal. The sender transaction would have to wait after writing each record to make sure that it got the correct response. Apart from using the link inefficiently, waiting between records would make the whole process impossibly slow. You can use distributed transaction processing to solve this problem, and others.

The flexibility of DTP can, in some circumstances, be used to achieve improved performance over function shipping. Consider browsing a remote file to select a record that satisfies some criteria. If you use function shipping, CICS ships the GETNEXT request across the link, and lets the mirror perform the operation and ship the record back to the requester. This is a lot of activity (two flows on the network) and the data flow can be significant. If the browse is on a large file, the overhead can be unacceptably high.

One alternative is to write a DTP conversation that ships the selection criteria, and returns only the keys and relevant fields from the selected records. This reduces both the number of flows and the amount of data sent over the link, thus reducing the overhead incurred in the function-shipping case.

Why distributed transaction processing?

In a multisystem environment, data transfers between systems are necessary because users need access to remote resources.

In managing these resources, network resources are used. But performance suffers if the network is used excessively. There is therefore a performance gain if application design is oriented toward doing the processing associated with a resource in the resource-owning region.

DTP lets you process data at the point where it arises, instead of overworking network resources by assembling it at a central processing point.

There are, of course, other reasons for using DTP. DTP does the following:

- Allows some measure of parallel processing to shorten response times
- Provides a common interface to a transaction that is to be attached by several different transactions
- Enables communication with applications running on other systems, particularly on non-CICS systems
- Provides a buffer between a security-sensitive file or database and an application, so that no application need know the format of the file records
- Enables batching of less urgent data destined for a remote system.

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 ONC Remote Procedure Calls
- The web interface.

This information discusses only distributed transaction processing. The other basic intercommunication facilities are described in Intercommunication methods.

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 section introduces you to the basic concepts involved in creating DTP applications. For a broad discussion of intercommunication concepts, see Getting started with intercommunication.

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 39 on page 112.

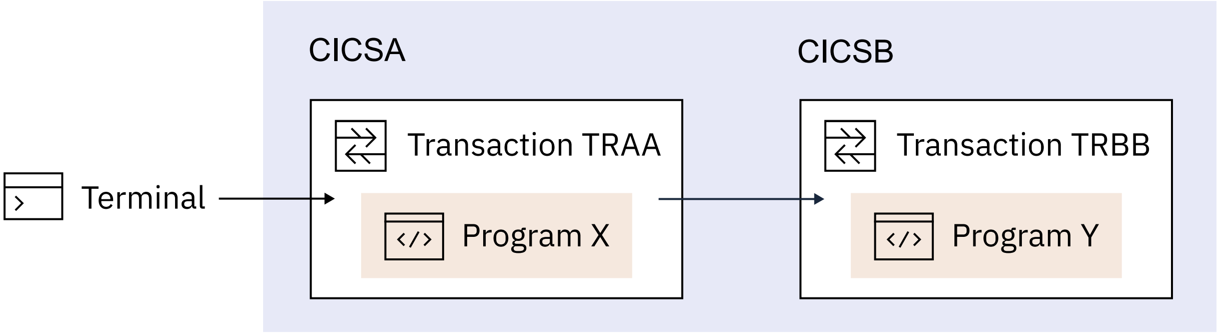


Figure 39. DTP between two CICS transactions

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 want 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 cancellation (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, 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 “Designing distributed processes” on page 118.

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 121 and “Maintaining data integrity” on page 117 respectively. Table 3 on page 114 shows which conversation states are defined for which protocols and synchronization levels.

Table 3. 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**.

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 40 on page 116 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.

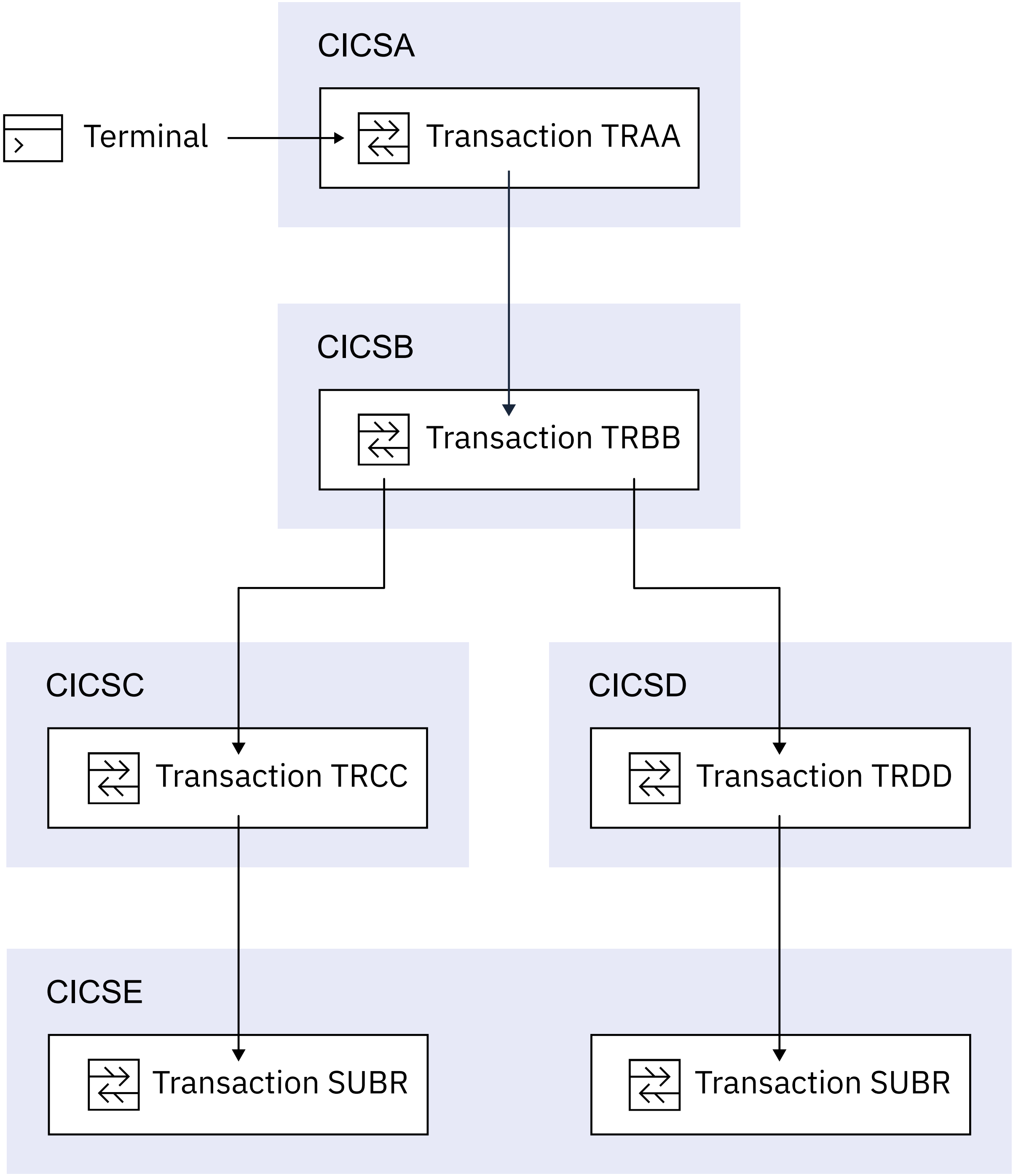


Figure 40. DTP in a distributed process

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.

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

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 might 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 function 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 *sync point*. Sync points are implied at the beginning and end of a transaction. A transaction can define other sync points by program command. All processing between two sync points 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 sync point command, CICS *commits* all changes to recoverable resources associated with that transaction. After the sync point, the transaction can no longer back out changes made since the previous sync point. 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.

The levels are:

- Level 0 – None
- Level 1 – Confirm
- Level 2 – Syncpoint

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

.

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.

Designing distributed processes

These topics discuss 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 What is a CICS application?.

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 must 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 might want to consider batching data locally before transmitting.

To maintain performance across the intersystem connection, the conversation must be freed as soon as possible — so that the session can 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. 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 might be appropriate. When input from the terminal user is required, terminate the the front-end transaction and its conversations. After the terminal user has responded, the successor front-end transaction can initiate a successor back-end transaction. If the first back-end transaction has 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 must 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. You can use the sysid of the principal facility and the identifier of the terminal to which the front-end transaction is attached.

Making maintenance easier

To correct errors or to adapt to the evolving needs of an organization, distributed processes inevitably have o 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. Using a single system 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 Security facilities in CICS.

Maintaining connectivity

If you require connectivity to transactions running in a back-level CICS system, check that the functions required are compatible in both systems.

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

Data conversion
For non-EBCDIC APPC logical units, some data conversion might be required on either receipt or sending of data.

Using multiple conversations
When using multiple, serial conversations, CICS might provide different conversation identifiers to the transaction. 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:

1. Transaction TRAA sends a 'last record' indicator to system B.

2. Transaction TRBB sends a response to system A.

There are several points at which you can 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.

Remember 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 might 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 API commands.

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 117 contains guidance on selecting the sync level for a conversation. Syncpointing a distributed process discusses the synchronization commands and their effects.

Selecting the protocol

CICS provides three different protocols that support distributed transaction processing. These protocols define the rules under which two transactions can communicate with each other.

The protocols are:

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

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 Migration of LUTYPE6.1 applications to APPC links.

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 Installation requirements for XCF/MRO.)

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 z/OS Communications Server overhead. So you must decide whether your application programs are to converse using APPC or MRO and code them accordingly.

Table 4 on page 121 points out the main differences between the MRO and APPC protocols.

Table 4. MRO protocol compared with APPC protocol

MRO	**APPC**
Function is realized without using a telecommunication access method.	Depends on z/OS Communications Server 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.

Table 4. MRO protocol compared with APPC protocol (continued)

MRO	APPC
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.

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.

Selecting the APPC conversation type

APPC conversations can be either *mapped* or *basic*. For CICS-to-CICS applications, you can use mapped conversations. Basic, or *unmapped*, conversations are useful to communicate with systems that do not support mapped conversations. These systems include some APPC devices.

The two protocols are similar. The main difference is the way that user data is formatted for transmission. In mapped conversations, the application sends the data that you want the partner to receive. In basic conversations, the application must include additional control bytes to convert the data to an SNA-defined format called a *generalized data stream* (GDS). Also, in EXEC CICS commands for basic conversations, you must include the keyword GDS.

The following table summarizes the differences between mapped and basic conversations that apply to the CICS API.

Table 5. APPC conversations – mapped or basic?

Mapped	Basic
The conversation partners exchange data that is relevant only to the application.	Both partners must package the user data before sending, and unpackage it on receipt.
All conversations for a transaction share the same EXEC Interface Block for status reporting.	Each conversation has its own area for state information.
The transaction can handle exception conditions or let them default.	The transaction must test for exception conditions in a data area set aside for the purpose.
A RECEIVE command issued in send state causes conversation turnaround.	A RECEIVE command is illegal in send state.
Transactions can be written in any of the supported languages.	Transactions can be written in assembler language or C only.
You can cause a conversation to time out if the partner does not respond. To do this, you specify the RTIMOUT option of the PROFILE definition.	You cannot cause a conversation to time out if the partner does not respond.

Effect of z/OS Communications Server persistent sessions support for DTP conversations on APPC sessions

If you enable z/OS Communications Server persistent sessions support in the local CICS, 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. DTP applications that use APPC sessions defined as persistent are affected by persistent sessions recovery.

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 persistent sessions recovery 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 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 Intersystem session queue management.
- 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 z/OS Communications Server persistent sessions support, see Recovery with z/OS Communications Server persistent sessions.

What is a conversation and what makes it necessary?

In DTP, transactions pass data to each other directly. While one sends, the other receives. The exchange of data between two transactions is called a **conversation**.

Although several transactions can be involved in a single distributed process, communication between them breaks down into a number of self-contained conversations between pairs. Each such conversation uses a CICS resource known as a **session**.

Conversation initiation and transaction hierarchy

A transaction starts a conversation by requesting the use of a session to a remote system. Having obtained the session, it causes an attach request to be sent to the other system to activate the transaction that is to be the conversation partner.

A transaction can initiate any number of other transactions, and hence, conversations. In a complex process, a distinct hierarchy emerges, with the terminal-initiated transaction at the very top. Figure 41 on page 124 shows a possible configuration. Transaction TRAA is attached over the terminal session. Transaction TRAA attaches transaction TRBB, which, in turn, attaches transactions TRCC and TRDD. Both these transactions attach the same transaction, SUBR, in system CICSE. This gives rise to two different tasks of SUBR.

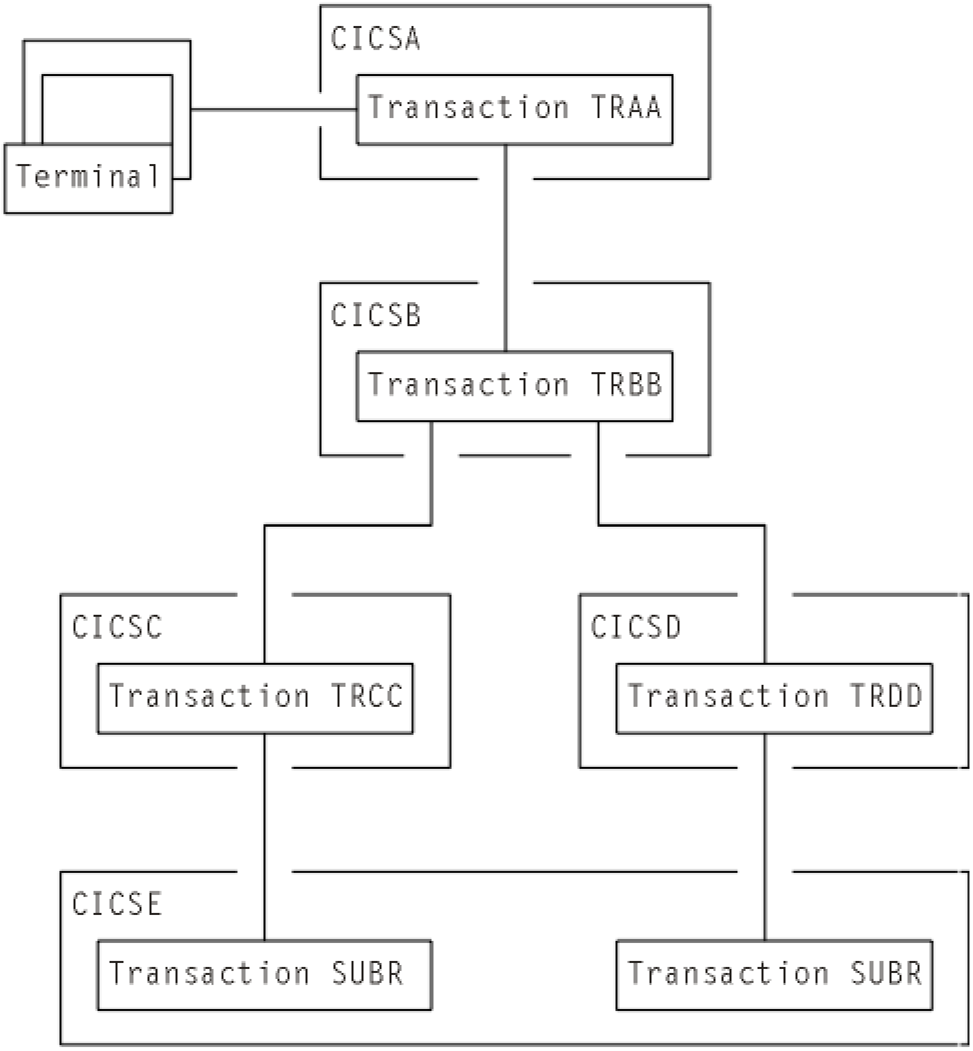


Figure 41. DTP in a multisystem configuration

The structure of a distributed process is determined dynamically by program; it cannot be predefined. Notice that, for every transaction, there is only one inbound attach request, but there can be any number of outbound attach requests. The session that activates a transaction is called its **principal facility**. A session that is allocated by a transaction to activate another transaction is called its **alternate facility**. Therefore, a transaction can have only one principal facility, but any number of alternate facilities.

When a transaction initiates a conversation, it is the **front end** on that conversation. Its conversation partner is the **back end** on the same conversation. (Some books refer to the front end as the initiator and the back end as the recipient.) It is normally the front end that dominates, and determines the way the conversation goes. You can arrange for the back end to take over if you want, but, in a complex process, this can cause unnecessary complication. This is further explained in the discussion on synchronization later in this chapter.

Dialog between two transactions

A conversation transfers data from one transaction to another.

For this to function properly, each transaction must know what the other intends. It would be nonsensical for the front end to send data if all the back end wants to do is print out the weekly sales report. It is therefore necessary to design, code, and test front end and back end as one software unit. The same applies when there are several conversations and several transaction programs. Each new conversation adds to the complexity of the overall design.

In the example in "Advantages over function shipping and transaction routing" on page 109, the DTP solution is to transmit the contents of the transient data queue from the front end to the back end. The front end issues a SEND command for each record that it takes off the queue. The back end issues RECEIVE commands until it receives an indication that the transmission has ended.

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

Control flows and brackets

During a conversation, data passes over the link in both directions.

A single transmission is called a **flow**. Issuing a SEND command does not always cause a flow. This is because the transmission of user data can be deferred; that is, held in a buffer until some event takes place. The APPC architecture defines data formats and packaging. CICS handles these things for you, and they concern you only if you need to trace flows for debugging.

The APPC architecture defines a data header for each transmission, which holds information about the purpose and structure of the data following. The header also contains bit indicators to convey control information to the other side. For example, if one side wants to tell the other that it can start sending, CICS sets a bit in the header that signals a change of direction in the conversation.

To keep flows to a minimum, non-urgent control indicators are accumulated until it is necessary to send user data, at which time they are added to the header.

For the formats of the headers and control indicators used by APPC, see Systems Network Architecture Formats (GA27-3136).

In complex procedures, such as establishing syncpoints, it is often necessary to send control indicators when there is no user data available to send. This is called a **control flow**.

begin_bracket marks the start of a conversation; that is, when a transaction is attached. conditional_end_bracket ends a conversation. End bracket is conditional because the conversation can be reopened under some circumstances. A conversation is **in bracket** when it is still active.

MRO is not unlike APPC in its internal organization. It is based on LUTYPE6.1, which is also an SNA-defined architecture.

Conversation state and error detection

As a conversation progresses, it moves from one state to another within both conversing transactions.

The conversation state determines the commands that may be issued. For example, it is no use trying to send or receive data if there is no session linking the front end to the back end. Similarly, if the back end signals end of conversation, the front end cannot receive any more data on the conversation.

Either end of the conversation can cause a change of state, usually by issuing a particular command from a particular state. CICS tracks these changes, and stops transactions from issuing the wrong command in the wrong state.

Synchronization

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

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

Examples of use

Suppose, for example, that a transaction is transmitting a queue of data to another system to be written to a DASD file. Suppose also that for some reason, not necessarily connected with the intercommunication activity, the receiving transaction is abended.

Even if a further abend can be prevented, there is the problem of how to continue the process without loss of data. It is uncertain how many queue items have been received and how many have been correctly written to the DASD file. The only safe way of continuing is to go back to a point where you know that the contents of the queue are consistent with the contents of the file. However, you then have two problems. On one side, you need to restore the queue entries that you have sent; on the other side, you need to delete the corresponding entries in the DASD file.

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

There are cases in which you may want to recover resources, even though there are no error conditions. Consider an order entry system. While entering an order for a customer, an operator is told by the system that the customer's credit limit would be exceeded if the order went through. Because there is no use continuing until the customer is consulted, the operator presses a function key to abandon the order. The transaction is programmed to respond by restoring the data resources to the state they were in at the start of the order.

Taking syncpoints

If you were to log your own data movements, you could arrange backout of your files and queues.

However, it would involve some very complex programming, which you would have to repeat for every similar application. To save you this overhead, CICS arranges resource recovery for you. LU management works with resource management in ensuring that resources can be restored.

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

Taking a syncpoint **commits** all recoverable resources. This means that all systems involved in a distributed process erase all the information they have been keeping about data movements on recoverable resources. Now backout is no longer possible, and all changes to the resources since the last syncpoint are made irreversible.

Although CICS commits and backs out changes to resources for you, the service must be paid for in performance. You might have transactions that do not need such complexity, and it would be wasteful to employ it. If the recovery of resources is not a problem, you can use simpler methods of synchronization.

The three sync levels

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

- Level 0 – none
- Level 1 – confirm
- Level 2 – syncpoint

At sync level 0, there is no system support for synchronization. It is nevertheless possible to achieve some degree of synchronization through the interchange of data, using the SEND and RECEIVE commands.

If you select sync level 1, you can use special commands for communication between the two conversation partners. One transaction can *confirm* the continued presence and readiness of the other. The user is responsible for preserving the data integrity of recoverable resources.

The level of synchronization described earlier in this section corresponds to sync level 2. Here, system support is available for maintaining the data integrity of recoverable resources.

CICS implies a syncpoint when it starts a transaction; that is, it initiates logging of changes to recoverable resources, but no control flows take place. CICS takes a full syncpoint when a transaction is normally terminated. Transaction abend causes rollback. The transactions themselves can initiate syncpoint or rollback requests. However, a syncpoint or rollback request is propagated to another transaction only when the originating transaction is in conversation with the other transaction, and if sync level 2 has been selected for the conversation between them.

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

MRO or APPC for DTP?

You can program DTP applications for both MRO and APPC links. The two conversation protocols are not identical. Although you seldom have the choice for a particular application, an awareness of the differences and similarities will help you to make decisions about compatibility.

Choosing between MRO and APPC can be quite simple. The options depend on the configuration of your CICS complex and on the nature of the conversation partner. You cannot use MRO to communicate with a partner in a non-CICS system. Further, it supports communication between transactions running in CICS systems in different MVS images only if the MVS images are in the same MVS sysplex, and are joined by cross-system coupling facility (XCF) links. For full details of the hardware and software requirements for XCF/MRO, see Installation requirements for XCF/MRO.

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

Table 6 on page 127 summarizes the main differences between the two protocols.

Table 6. MRO compared with APPC

MRO	APPC
Function is realized within CICS	Depends on the z/OS Communications Server or similar
Nonstandard architecture	SNA architecture
CICS-to-CICS links only	Links to non-CICS systems possible
Communicates within single MVS image, or (using XCF/MRO) between MVS images in same sysplex	Communicates across multiple MVS images and other operating systems
PIP data not supported	PIP data supported
Data transmission not deferred	Deferred data transmission
Partner transaction identified in data	Partner transaction defined by program command
RECEIVE can only be issued in receive state	RECEIVE causes conversation turnaround when issued in send state on mapped conversations
No expedited flow possible	ISSUE SIGNAL command flows expedited
WAIT command has no function	WAIT command causes transmission of deferred data

APPC mapped or basic?

APPC conversations can be either *mapped* or *basic*. For CICS-to-CICS applications, you can use mapped conversations. Basic, or *unmapped*, conversations are useful to communicate with systems that do not support mapped conversations. These systems include some APPC devices.

The two protocols are similar. The main difference is the way that user data is formatted for transmission. In mapped conversations, the application sends the data that you want the partner to receive. In basic conversations, the application must include additional control bytes to convert the data to an SNA-defined format called a *generalized data stream* (GDS). Also, in EXEC CICS commands for basic conversations, you must include the keyword GDS.

Table 7 on page 128 summarizes the differences between mapped and basic conversations that apply to the CICS API.

CPI Communications has different rules (see "EXEC CICS or CPI Communications?" on page 128.

Table 7. APPC conversations – mapped or basic?

Mapped	Basic
The conversation partners exchange data that is relevant only to the application.	Both partners must package the user data before sending, and unpackage it on receipt.
All conversations for a transaction share the same EXEC Interface Block for status reporting.	Each conversation has its own area for state information.
The transaction can handle exception conditions or let them default.	The transaction must test for exception conditions in a data area set aside for the purpose.
A RECEIVE command issued in send state causes conversation turnaround.	A RECEIVE command is illegal in send state.
Transactions can be written in any of the supported languages.	Transactions can be written in assembler language or C only.
You can cause a conversation to time out if the partner does not respond. To do this, you specify the RTIMOUT option of the PROFILE definition.	You cannot cause a conversation to time out if the partner does not respond.

EXEC CICS or CPI Communications?

CICS gives you a choice of two application programming interfaces (APIs) for coding your DTP conversations on APPC sessions.

The first, the **CICS API**, is the programming interface of the CICS implementation of the APPC architecture. It consists of EXEC CICS commands and can be used with all CICS-supported languages. The second, **Common Programming Interface Communications** (CPI Communications) is the communication interface defined for the SAA environment. It consists of a set of defined verbs, in the form of program calls, which are adapted for the language being used.

Table 8 on page 128 compares the two methods to help you to decide which API to use for a particular application.

Table 8. CICS API compared with CPI Communications

CICS API	CPI Communications
Portability between different members of the CICS family.	Portability between systems that support SAA facilities.
Basic conversations can be programmed only in assembler language or C.	Basic conversations can be programmed in any of the available languages.

Table 8. CICS API compared with CPI Communications (continued)

CICS API	CPI Communications
Sync levels 0, 1, and 2 supported.	Sync levels 0, 1, and 2 supported, *except for transaction routing, for which only sync levels 0 and 1 are supported*.
PIP data supported.	PIP data not supported.
Only a few conversation characteristics are programmable. The rest are defined by resource definition.	Most conversation characteristics can be changed dynamically by the transaction program.
Can be used on the principal facility to a transaction started by ATI.	Cannot be used on the principal facility to a transaction started by ATI.
Limited compatibility with MRO.	No compatibility with MRO.

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

Introduction to IP interconnectivity

IP interconnectivity (IPIC) is a type of intercommunication link that enables you to integrate CICS-to-CICS communications into an IP infrastructure and use the secure sockets layer (SSL) to provide security.

IPIC uses the same flow of data and control from the local program to the remote program as ISC. IPIC also supports two-phase commit, as well as channels and containers.

IPIC supports both IPv4 and IPv6 TCP/IP protocols. For more information about configuring IPv4 or IPv6 addressing, see Configuring IP interconnectivity.

IPIC supports the following types of intercommunication functions for their respective product releases:

- Distributed program link (DPL) calls between CICS regions and between CICS TS and TXSeries.
- Asynchronous processing of **EXEC CICS START**, **START CHANNEL**, and **CANCEL** commands between CICS regions.
- Transaction routing of 3270 terminals, where the terminal-owning region (TOR) is uniquely identified by an APPLID between CICS regions.
- Enhanced method of routing transactions that are invoked by **EXEC CICS START** commands between CICS regions.
- ECI requests from CICS Transaction Gateway.
- Function shipping of all file control, transient data, and temporary storage requests between CICS regions. Function shipping using IPIC connectivity is threadsafe between CICS regions at a supported release level.
- Threadsafe processing for the mirror program and the LINK command to improve performance for threadsafe applications.
- RESTful API calls through z/OS Connect Enterprise Edition

IPIC resources

An IPIC connection requires two related CICS resources in each pair of CICS regions: an IPCONN resource and a TCPIPSERVICE resource.

- The IPCONN resource defines the type of intercommunication connection.
- The TCPIPSERVICE resource defines which protocol will be used for this TCP/IP service (IPIC) and the port that is used to listen on.

The TCPIPSERVICE accepts messages arriving on its port which conform to IPIC protocol standards. Any messages arriving at the port that do not conform to this format are rejected.

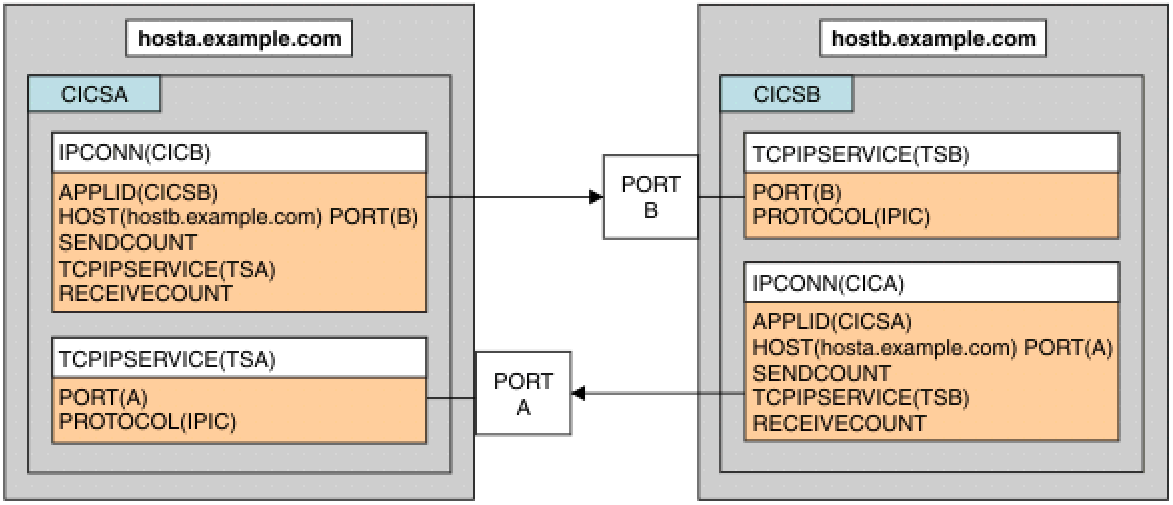
You can either create or autoinstall new IPIC connections or you can migrate your existing APPC and MRO connections.

Typical scenario

The following simple scenario shows how you can configure two CICS regions to create IPIC connections between them.

In the scenario, the two CICS regions are on separate MVS images. The CICS region called CICSA runs on the hosta.example.com image. It uses TCP/IP to connect to the other CICS region, CICSB, which is on a different MVS image.

This figure shows the connections between the CICS regions in the scenario and the resource definitions that are required in each region to establish the IPIC connection.



Prerequisites for IPIC

To install and use IPIC for your CICS applications, you must have CICS regions of the correct release level, and you must have access to a TCP/IP network.

The system requirements are as follows:

- At least two CICS regions that are at CICS TS for z/OS, Version 3.2 or higher. IPIC is not available in earlier releases of CICS.
- TCP/IP services must be active in the CICS regions. You can activate these services by setting the **TCPIP** and **ISC** system initialization parameters to YES.
- Each CICS region must have access to a TCP/IP stack running on the same MVS image.
- The TCP/IP network must extend between these images when each CICS region exists on a different MVS image.
- Review your **MAXSOCKETS** system initialization parameter settings. Ensure that you allocate enough sockets to support IPIC connections and other traffic that requires IP sockets.

Chapter 2. Configuring CICS interconnectivity

Configuration requirements are different, depending on whether you want to use intersystem communication to connect regions using SNA or use multiregion operation (MRO). You can configure CICS to communicate over TCP/IP or over SNA in an intersystem communication environment.

"Configuring support for communicating over a TCP/IP network" on page 131 describes how to set up set up TCP/IP services to use a number of CICS-supported protocols, including HTTP and IPIC.

"Configuring support for ISC over SNA" on page 132 provides guidance on ACF/Communications Server and IMS.

"Steps after configuring MRO" on page 132 describes how to set up CICS for multiregion operation.

"Configuring z/OS Communications Server generic resources" on page 133 describes how to register your terminal-owning regions as members of a VTAM® generic resource group, and things you need to consider when doing so.

Configuring support for communicating over a TCP/IP network

CICS operating in a dual-mode environment uses both IPv4 and IPv6 networks and always attempts to communicate using IPv6 before using the IPv4 network. A single-mode environment operates in an IPv4 network only. You can set up TCP/IP services to use a number of CICS-supported protocols, including HTTP and IPIC.

Before you begin

You need a minimum level of CICS TS 4.1 to communicate using IPv6. The CICS region must be running in a dual-mode (IPv4 and IPv6) environment and the client or server with which CICS is communicating must also be running in a dual-mode environment. If a region is running in a single-mode (IPv4) environment or a region is operating at a pre-CICS TS 4.1 release, you can communicate using IPv4 only.

About this task

Follow these steps to configure your connection to use either IPv4 or IPv6 addressing, or a combination of the two formats.

Procedure

1. Activate TCP/IP services by specifying **TCPIP=YES** as a system initialization parameter.
2. Define resources to support the protocol you are using to communicate over in the TCP/IP network. Here are examples of two different protocols which can be defined using resources:
   a) If you are using IPIC, define and install a TCPIPSERVICE resource and an IPCONN resource in both partner regions.

      See "Defining IP interconnectivity (IPIC) connections" on page 155 for examples and instructions to help you define and install your resource definitions.

   b) If you are using HTTP with CICS as an HTTP client, define and install a URIMAP resource in the issuing region and a TCPIPSERVICE resource in the listening region. Define the host name, IPv4 or IPv6 address that you want to use in the HOST attribute of the URIMAP(CLIENT) resource definition.

      See Creating a URIMAP resource for CICS as a HTTP client for information about URIMAP resources for HTTP requests.
3. Optional: Advise your network administrator to define an IPv4 primary interface address to ensure that you do not have problems when communicating outside of a CICSplex. The primary interface address is the address that is specified in the PRIMARYINTERFACE statement for the TCPIP.PROFILE.

If you issue a **GETHOSTID** call, **GETHOSTID** returns the IPv4 primary interface address, or the loopback address if **GETHOSTID** cannot find a host address. The **IPRESOLVED** option stores the address returned by **GETHOSTID**, so **IPRESOLVED** might contain either the primary interface address, or the loopback address. If you are communicating outside of the CICSplex, results can be unpredictable if a loopback address is returned. To define a primary interface address, see the information about the TCP/IP address space, PROFILE.TCPIP, in the z/OS Communications Server: IP Configuration Guide.

Results

The TCP/IP connection is correctly configured and is available for use over an IPv4 connection.

Your connection will also be available over IPv6 if you have the correct level of CICS and your environments have dual-mode capability.

What to do next

If you are having problems with your connection, see Dealing with TCP/IP connectivity problems.

Configuring support for ISC over SNA

The information on ACF/Communications Server and IMS given in this section is for guidance only. Always consult the current ACF/Communications Server or IMS publications for the latest information. ISC over SNA uses the ACF/Communications Server access method, so when you install ACF/Communications Server, you must include intersystem communication programs and operands in your system to allow intersystem communication over SNA (ISC over SNA).

1. Include the intersystem communication programs in your system by specifying YES on the z/OS Communications Server and ISC system initialization parameters.
2. When you define your CICS system to ACF/Communications Server, include intersystem communication operands in the z/OS Communications Server APPL statement.
3. If your CICS installation is to use CICS-to-IMS intersystem communication, ensure that the CICS and the IMS installations are fully compatible. For more information about defining compatible CICS and IMS nodes, see “Defining connections to remote systems” on page 153. For full details of IMS installation, see Installation in IMS product documentation.
   a. Include intersystem communication operands in the z/OS Communications Server APPL statement.
   b. Define IMS ISC-related macros and parameters. See “Defining compatible CICS and IMS nodes” on page 182.

Steps after configuring MRO

When you have configured MRO support, you must define the MRO connection and resources.

Procedure

1. Define MRO connection to the remote systems. For more information, see “Defining links for multiregion operation” on page 170.
2. Define resources on both the local CICS region and remote systems. For more information, see “Defining local resources” on page 218 and “Defining remote resources” on page 200.

Configuring z/OS Communications Server generic resources

In a CICSplex containing a set of functionally-equivalent CICS terminal-owning regions (TORs), you can use the z/OS Communications Server generic resource function to balance terminal sessions across the available TORs.

This topic assumes some knowledge of tasks, such as defining connections to remote systems. For information on defining links to remote systems, see "Defining connections to remote systems" on page 153.

For an overview of Communications Server generic resources, see Workload balancing in a sysplex.

This section contains the following topics:

Prerequisites for z/OS Communications Server generic resources

To use z/OS Communications Server generic resources, you need ACF/Communications Server Version 4 Release 2 or a later, upward-compatible, release.

- z/OS Communications Server must be running under an MVS environment that is part of a sysplex.
- z/OS Communications Server must be connected to the sysplex coupling facility. For information about the sysplex coupling facility, see z/OS MVS Setting Up a Sysplex.
- At least one z/OS Communications Server in the sysplex must be an advanced peer-to-peer networking (APPN) network node, with the other z/OS Communications Servers being APPN end nodes.

Planning your CICSplex to use z/OS Communications Server generic resources

You can use the z/OS Communications Server generic resource function to balance terminal session workload across a number of CICS regions.

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

APPC and LUTYPE6.1 connections do not log on in the same way as terminals. But they too can establish a connection to a generic resource by using either the generic resource name (in which case z/OS

Communications Server chooses the member to which the connection is made) or the member name (in which case the connection is made to the named member).

When you plan your CICSplex to use z/OS Communications Server generic resources, you need to consider the following:

- Which CICS regions should be generic resource members?

  Note that:

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

  If you have several groups of users who use different applications, you may want to set up several generic resources, one for each group of users. Bear in mind that a single CICS region cannot be a member of more than one generic resource at a time.
- Will there be APPC or LUTYPE6.1 connections. You are recommended to use APPC in preference to LUTYPE6.1 for CICS-to-CICS connections:

  - Between members of a generic resource? You cannot use LUTYPE6.1 connections between members of a generic resource.
  - Between members of one generic resource and members of another generic resource?
  - Between members of a generic resource and systems which are not members of generic resources?

  In all these cases you will need to understand when you can use:

  - Connection definitions that specify the generic resource name of the partner system
  - Connection definitions that specify the member name of the partner system
  - Autoinstall to provide definitions of the partner system.

Naming the CICS regions

Every CICS region has a network name, defined on a z/OS Communications Server APPL statement, that uniquely identifies it to z/OS Communications Server.

You specify this name, or *applid*, on the APPLID system initialization parameter. If a region is a member of a generic resource, its applid and member name are one and the same.

A generic resource—a collection of CICS regions—has a generic resource name. Each CICS region that is to be a member of a generic resource specifies the generic resource name on its GRNAME system initialization parameter. Unlike network names, generic resource names do not have to be defined to z/OS Communications Server. However, they must be distinct from network names, and must be unique within a network.

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

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

Defining connections in a generic resource environment

The z/OS Communications Server generic resource function can be used to balance session workload for APPC and LUTYPE6.1 connections.

Connections differ from terminal sessions in the following ways:

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

Defining connections

When you define a connection to a generic resource, you have two possibilities for the NETNAME attribute of the CONNECTION resource.

About this task

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

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

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

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

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

Defining connections between GR members and non-GR members

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

Defining connections between members within a generic resource

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

Defining connections between CICS TS for z/OS generic resources

If you have two CICS TS for z/OS generic resources, you do not need to define and install member name connections for every possible connection between them.

Instead, you can define and install a single generic resource name connection in each member that may initiate a connection with the partner generic resource. CICS then autoinstalls member name connections as they are required.

The only connection definition required in a CICS region that does not initiate connections is one that can be used as an autoinstall template. If there is a generic resource name connection installed, it is used as the template, so we suggest that you define generic resource name connections for this purpose.

Generating z/OS Communications Server generic resource support

To generate z/OS Communications Server generic resource support for your CICS TORs, you must perform these steps.

About this task

If your CICSplex comprises separate terminal-owning regions and application-owning regions, do not include TORs and AORs in the same generic resource group.

Procedure

1. Use the GRNAME system initialization parameter to define the generic resource name under which CICS is to register to z/OS Communications Server. To comply with the CICS naming conventions, pad the name to the permitted 8 characters with one of the characters #, @, or $.
   For example:

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

   If you specify a valid generic resource name on **GRNAME**, specify only *name1* on the **APPLID** system initialization parameter. If you do specify both *name1* and *name2* on the **APPLID** parameter, CICS ignores *name1* and uses *name2* as the z/OS Communications Server APPLID.
2. Use an APPL statement to define the attributes of each participating TOR to z/OS Communications Server.

   The attributes defined on each individual APPL statement should be identical. The name on each APPL statement must be unique. It identifies the TOR individually, within the generic resource group.
3. Shut down each terminal-owning region normally before registering it as a member of the generic resource.

   An immediate shutdown is *not* sufficient; nor is a CICS failure followed by a cold start. Do not specify a shutdown assist transaction, to avoid the possibility of the transaction force closing z/OS Communications Server or performing an immediate shutdown. The default shutdown assist transaction, DFHCESD, is described in Shutdown assist program (DFHCESD).

   If CICS has *not* been shut down cleanly before you try to register it as a member of a generic resource, z/OS Communications Server might (due to the existence of persistent sessions) fail to register it, and issue a return code-feedback (RTNCD-FDB2) of X'14', X'86'. To correct this, you must restart CICS (with the same APPLID), and then shut it down cleanly. Alternatively, if you have written a batch program to end affinities (see "Writing a batch program to end affinities" on page 145), you might be able to use it to achieve the same effect. As part of its processing, the batch program opens the original z/OS Communications Server ACB with the original APPLID, unbinds any persisting sessions, and closes the ACB.

Migrating a TOR to a generic resource

This section describes how to manage existing terminals and connections when migrating a TOR to membership of a CICS Transaction Server for z/OS generic resource.

How to establish connections between two CICS TS for z/OS generic resources is described separately in "Setting up inter-sysplex communications between generic resources" on page 139.

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

Recommended methods

For simplicity, first create a generic resource consisting of only one member. Do not add further members until the single-member generic resource is functioning satisfactorily.

Because all members of a generic resource should be functionally equivalent, you create additional members by cloning the first member. (A situation in which you might choose to ignore this advice is described in this document.)

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

No LU6 connections

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

About this task

You can then gradually migrate the terminals to using the generic resource name. Later, you can expand the generic resource by cloning the first member-TOR.

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

LU6 connections

If there are LU6 (APPC or LU6.1) connections to your terminal-owning region, not counting connections to other members of the generic resource, we recommend that they log on using the generic resource name. However, you will probably want to migrate to generic resource without requiring all your LU6 network partners to change their logon procedures.

About this task

One option is to use the applid of your existing terminal-owning region as the new generic resource name. Because this requires you to choose a new applid, it is also necessary to change the CONNECTION definitions of MRO-connected application-owning regions and RACF profiles that specify the old applid. Note, however, that you do not need to change the APPL profile to which the users are authorized—CICS passes the GRNAME to RACF as the APPL name during signon validation, and the old applid is now the GRNAME. The recommended migration steps are:

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

6. Restart the CICSplex.

   At this point:

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

   At this point:

   - Autoinstalled non-LU6 terminals start to exploit session balancing.
   - Autoinstalled APPC sync level 1 connections start to exploit session balancing.
   - Because of affinities, existing LU6.1 and APPC sync level 2 connections continue to be connected to the original terminal-owning region (by generic resource name).
   - Special considerations apply to non-autoinstalled terminals and connections, and to LU6 connections used for outbound requests. These are described in "Dealing with special cases" on page 150.

Removing a TOR from a generic resource

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

About this task

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

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

  To re-register CICS with the generic resource, you must close and reopen the z/OS Communications Server ACB.

Important:

If you remove a region from a generic resource:

- You should end any affinities that it owns. If you do not, z/OS Communications Server will not allow the affected APPC and LU6.1 partners to connect to other members of the generic resource. See "Ending affinities" on page 143.
- The region that has been removed should not try to acquire a connection to a partner that knows it by its generic resource name, unless the partner has ended its affinity to the removed region.

Moving a TOR to a different generic resource

To move a region from one generic resource to another, you must perform the following steps.

About this task

1. End any affinities that it owns. See "Ending affinities" on page 143.

2. Shut it down cleanly. See "Generating z/OS Communications Server generic resource support" on page 136.

   If CICS is *not* shut down cleanly before you try to register it as a member of the new generic resource, z/OS Communications Server may fail to register it, and issue a RTNCD-FDB2 of X'14', X'86'. To correct this, you must restart CICS with the *original* GRNAME and APPLID, then shut it down normally. Do not specify a shutdown assist transaction, to avoid the possibility of the transaction force closing z/OS Communications Server or performing an immediate shutdown.

   Alternatively, if you have written a batch program to end affinities, you might be able to use it to achieve the same effect. As part of its processing, the skeleton program described in "Writing a batch program to end affinities" on page 145 opens the original z/OS Communications Server ACB with the original GRNAME, unbinds any persisting sessions, and closes the ACB.
3. Specify the name of the alternative generic resource on the GRNAME system initialization parameter, and restart CICS.

Setting up inter-sysplex communications between generic resources

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

Establishing connections between CICS TS for z/OS generic resources

Assume that you have two sysplexes, SYSPLEXL and SYSPLEXR, and that these contain the CICS TS for z/OS generic resource groups CICSL and CICSR, respectively.

About this task

This is illustrated by Figure 42 on page 140. The steps involved in establishing connections between CICSL and CICSR are as follows:

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

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

   The first attempt by any member of CICSL to acquire a connection to CICSR (or vice versa) uses a generic resource name connection.
2. The CICSR member to which z/OS Communications Server sends the bind request searches for the generic resource name connection definition for CICSL. (If none exists, it autoinstalls one, subject to the normal rules for autoinstalling connections.)
3. Subsequent connections that z/OS Communications Server happens to route to the same member of CICSR from different members of CICSL are autoinstalled on the CICSR member, using the CICSL member name as the NETNAME; that is, CICS autoinstalls *member name connections*. Similarly, subsequent connections to the same member of CICSL from different members of CICSR are autoinstalled on the CICSL member, using the CICSR member name as the NETNAME. The example in "Example" on page 140 makes this clearer.

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

   If you decide to use a template other than the default for member name connections, remember that use of the sessions for these connections is initiated by the partner, so consider defining the MAXIMUM attribute of the SESSIONS resource with no contention winners. This attribute is described in "Defining groups of APPC sessions" on page 176. This is useful because the member name is not known to the applications in the system in which the member name connection is autoinstalled.

They use the GR name for outbound requests. Therefore the member name connection is not used for outbound requests and so does not need to have any sessions defined as winners. By allowing the partner system to have all the sessions as winners, the overhead of bidding for loser sessions is avoided.

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

Example

An example of establishing connections between CICS TS for z/OS generic resources.

In Figure 42 on page 140 through Figure 45 on page 143, each generic resource uses the partner sysplex's generic resource name when initiating a connection. All generic resource members are able to initiate connections; that is, they all have a generic resource name connection (a predefined connection entry in which the NETNAME is the generic resource name of the partner sysplex). The connections are APPC parallel-session synclevel 2 links.

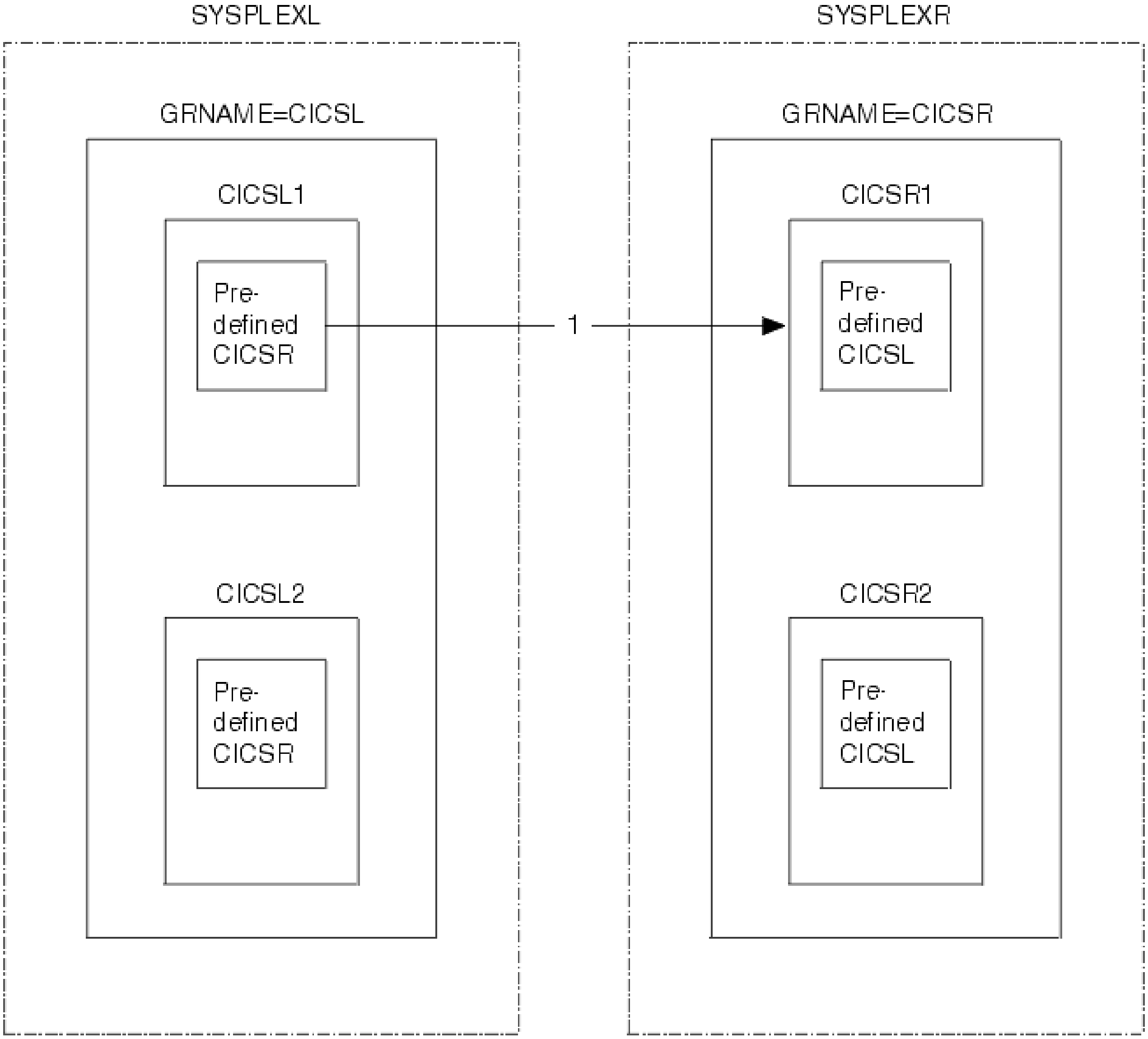


Figure 42. The figure shows two sysplexes, SYSPLEXL and SYSPLEXR

In Figure 42 on page 140, the first bind that flows from CICSL1 to CICSR is routed to whichever member of CICSR z/OS Communications Server decides is the most lightly loaded. In this example it goes to

CICSR1. The predefined connections for the generic resource names CICSR and CICSL in CICSL1 and CICSR1 are used.

Affinities are created at SYSPLEXL and SYSPLEXR, associating CICSL1 with CICSR1. When you need to end these affinities, you may or may not need to do so explicitly—see "Ending affinities" on page 143 and APPC connection quiesce processing. Until the affinities are ended, whenever CICSL1 tries to reconnect to CICSR, z/OS Communications Server routes the request to CICSR1; and whenever CICSR1 tries to reconnect to CICSL, z/OS Communications Server routes the request to CICSL1.

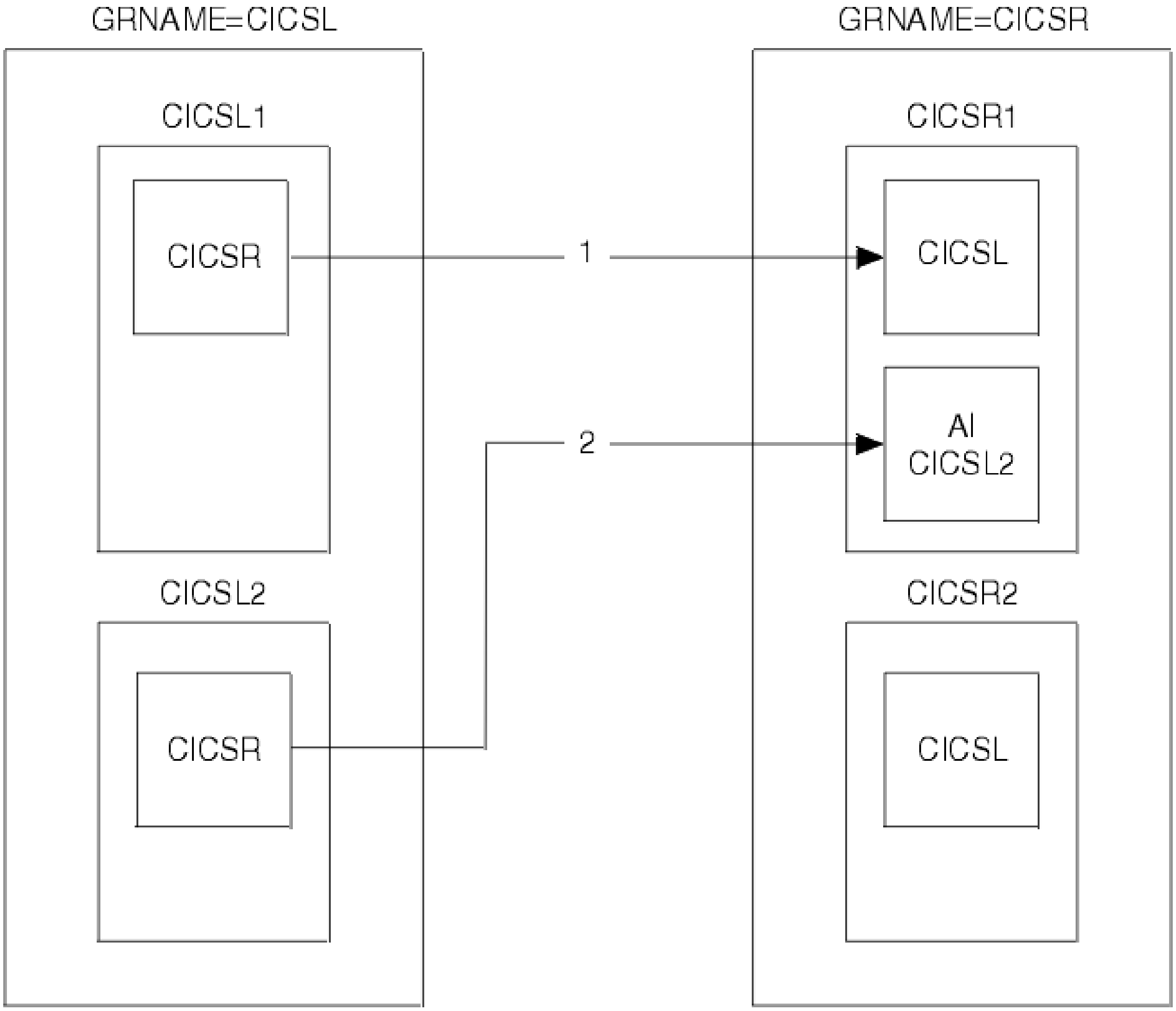


Figure 43. Second flow, CICSL2-CICSR

Figure 43 on page 141 shows a bind flow from CICSL2 to CICSR. In this example z/OS Communications Server has, once again, chosen to route it to CICSR1, but it could have gone to one of the other members of CICSR.

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

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

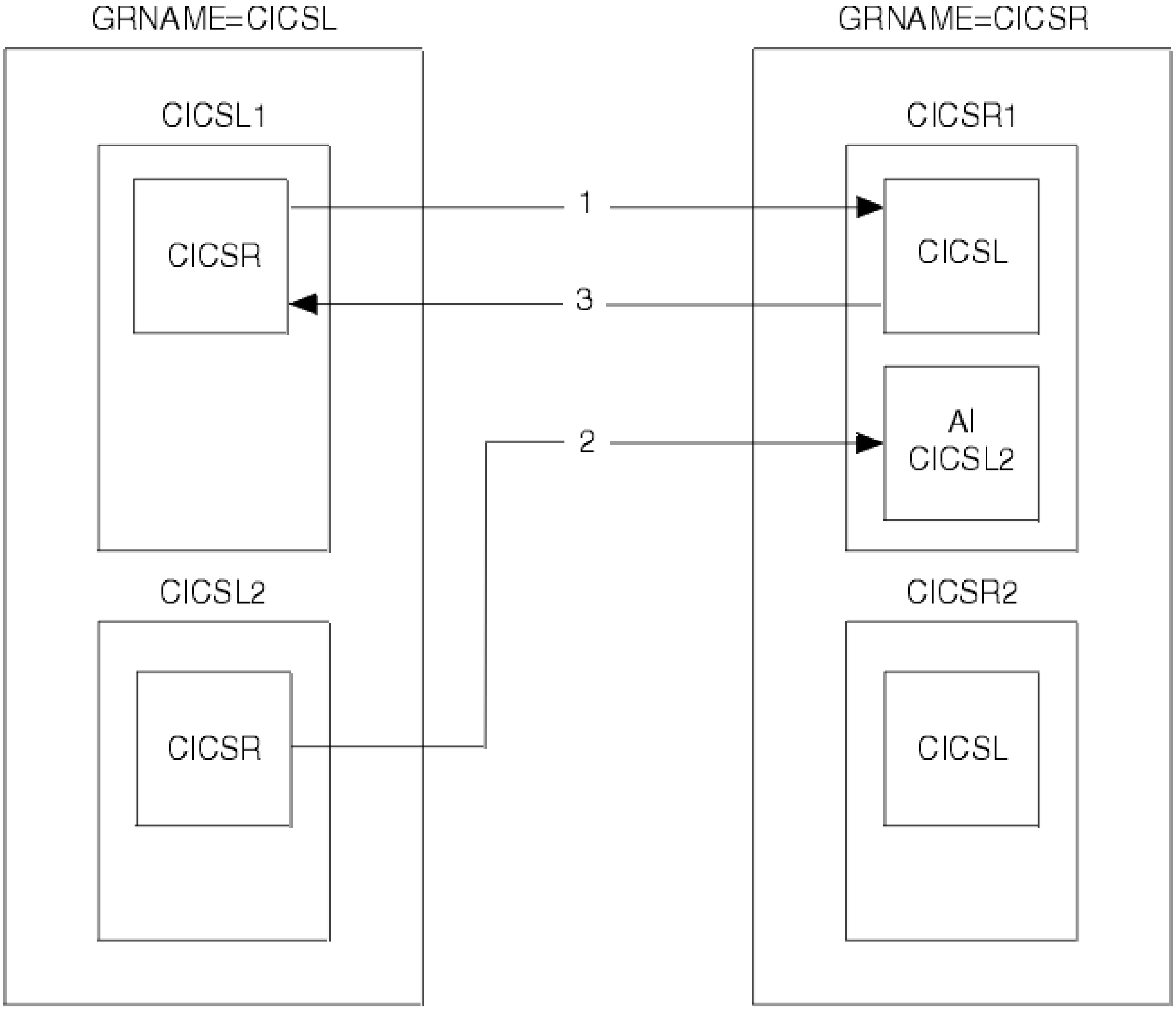


Figure 44. Third flow, CICSR1-CICSL

Figure 44 on page 142 shows a third flow, this time from CICSR1 to CICSL. The existing affinity forces it to CICSL1.

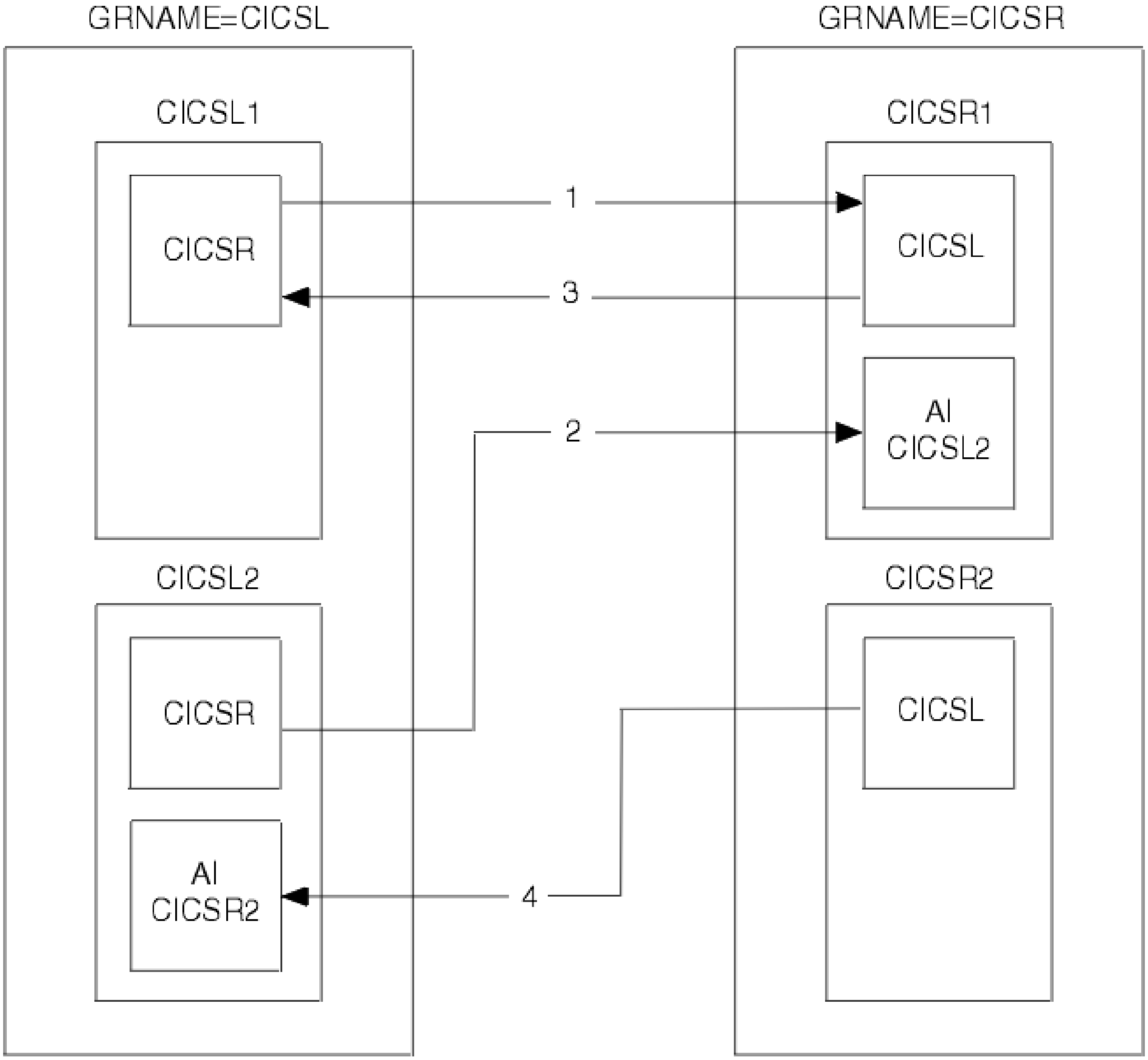


Figure 45. Fourth flow, CICSR2-CICSL

Figure 45 on page 143 shows a fourth flow, this time from CICSR2 to CICSL. It can go to any member of CICSL, but in this example z/OS Communications Server routes it to CICSL2.

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

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

Ending affinities

When a session is established with a member of a generic resource, z/OS Communications Server creates an association called an affinity between the generic resource member and the partner LU, so that it knows where to route subsequent flows.

In most cases, z/OS Communications Server ends the affinity when all activity on the session has ceased. However, for some types of session, z/OS Communications Server assumes that resynchronization data might be present, and therefore relies on CICS to end the affinity. The following sessions are affected:

- APPC synclevel 2 sessions

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

In z/OS Communications Server terms, the CICS generic resource member "owns" the affinity and is responsible for ending it. The affinity persists even after a connection is deleted or CICS has performed an initial or cold start. *For a connection between two generic resources, both partners own an affinity, and each must be ended*. For APPC connections between regions, the APPC connection quiesce protocol does this automatically; see APPC connection quiesce processing. For other connections, the affinities must be ended explicitly.

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

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

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

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

Having received message DFHZC0177, to check whether an affinity that must be ended explicitly does exist, you can use the SNA **D NET,GRAFFIN** command. This command produces messages IST1706 and IST1707, which should contain the information you need. Alternatively, produce a dump of the z/OS Communications Server ISTGENERIC data area. This dump contains SPTE records that show which affinities exist. For more information, see z/OS MVS IPCS Commands. For example, use the following command to start the dump:

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

Reply with the following command:

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

Use the following command to look at the dump:

```
STRDATA DETAIL ALLSTRS ALLDATA
```

If z/OS Communications Server rejects a request to end an affinity because no such affinity exists, message DFHZC0181 is issued. This can mean either that you supplied an incorrect NETNAME or NETID, or that the possible affinity does not actually exist.

When should you end affinities?

You need to end affinities if you reconfigure your sysplex.

For example, you **must** end any relevant affinities before you do any of the following:

- Change the name of a generic resource.
- Change a generic resource name connection to a member-name connection.
- Change a parallel-session connection to a single-session connection.

- Remove systems from a generic resource. If you remove a system from a generic resource and do not end its affinities, z/OS Communications Server treats it as though it were still a member of the generic resource.

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

Writing a batch program to end affinities

If a generic resource member that owns affinities fails and cannot be recovered, the affinities must be ended.

Important: Use this technique only if it is impossible to restart the failed CICS system.

In this situation, you cannot use the **`SET CONNECTION ENDAFFINITY`** or **`PERFORM ENDAFFINITY`** commands. Instead, you can use a batch program to clear the affinities owned by the failed member. The batch program must be written in assembler language. You can use the dump technique described in z/OS MVS IPCS Commands to find out which affinities the failed generic resource member owns.

Program input

You need to specify the following input parameters to the program.

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

Program output

The program uses the z/OS Communications Server CHANGE OPTCD=ENDAFFIN macro to end the affinities.

You probably need to produce a report on the success or failure of this and the other Communications Server macro calls that the program uses. See z/OS Communications Server: SNA Programming for the meaning of RTNCD/FDB2 values.

Processing

The processing that the program needs to perform is listed.

About this task

Be aware of the following programming notes:

- The z/OS Communications Server commands should be synchronous, to avoid the use of exits (OPTCD=SYN).
- Take care **not** to run the program for an APPLID of a running CICS. If you do, and you are using z/OS Communications Server persistent sessions, a *predatory takeover* will occur; that is, your program will assume control of the sessions belonging to the APPLID.

VTAM is the previous name for z/OS Communications Server

Programming notes:

1. The z/OS Communications Server commands should be synchronous, to avoid the use of exits (OPTCD=SYN).
2. Take care **not** to run the program for an APPLID of a running CICS. If you do, and you are using z/OS Communications Server persistent sessions, a *predatory takeover* will occur; that is, your program will assume control of the sessions belonging to the APPLID.

Procedure

1. Reserve storage for the following:

- The ACB of the failed sysplex member. The following example assumes that you are using persistent sessions:

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

- The RPL, which is required by the z/OS Communications Server macros:

  ```
  rpl-name RPL   AM=VTAM,OPTCD=(SYN)
  ```

- The NIB, which is required by the CHANGE OPTCD=ENDAFFIN macro:

  ```
  nib-name NIB
  ```

2. Issue a VTAM OPEN command for the ACB of the member that owns the affinity, passing the input APPLID for this member.
3. If any sessions persist, use the VTAM SENDCMD macro to terminate them. If you are not using persistent sessions, this is not necessary.

   a. Move the following command to an area in storage. In this example, *applid1* is the member name of the failed member and *applid2* is the APPLID of the partner system.

   ```
   'VARY NET,TERM,LU1=applid1,LU2=applid2,TYPE=FORCE,SCOPE=ALL'
   ```

   b. Issue the SENDCMD macro, as in the following example. In this example:

   - *rpl-name* is the name of an RPL.
   - *acb-name* is the ACB of the failed sysplex member.
   - *output-area* is the name an area in storage where the VARY command is held.
   - *command-length* is the length of the command.

   ```
   SENDCMD  RPL=rpl-name,
      ACB=acb-name,
      AREA=output-area,
      RECLEN=command-length,
      OPTCD=(SYN)
   ```

4. Use the VTAM RCVCMD macro to receive messages from z/OS Communications Server.

   RCVCMD must be issued three times after the SENDCMD to be sure that the VARY command worked correctly. In the following example:

   - *rpl-name* and *acb-name* are as described previously.
   - *input-area* is the area of storage into which the message is to be received.
   - *receive_length* is the length of data to be received.

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

5. Issue this command twice more to make sure of receiving all the output from z/OS Communications Server.
6. Issue the VTAM CHANGE OPTCD=ENDAFFIN macro to end the affinity.

   Before issuing the macro, the following fields must be initialized in the NIB:

   - NIBSYM is set to the APPLID of the partner system.
   - NIBGENN is set to the generic resource name of the failed system.
   - NIBNET is set to the NETID of the partner system.

   ```
   CHANGE  RPL=rpl-name,
           ACB=acb-name,
   ```

```
         NIB=nib-name,
         OPTCD=(SYN,ENDAFFIN)
```

7. Issue the VTAM CLOSE command for the ACB.

JCL for submitting the ENDAFFINITY program

This is an example of JCL for submitting the ENDAFFINITY program.

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

Figure 46. Example JCL for submitting the ENDAFFINITY program

Using ATI with generic resources

Automatic transaction initiation (ATI) is the process whereby a transaction is started by a request made internally within the CICS system, rather than by a terminal end-user entering a transaction name.

This can happen when, for example, an application program issues an EXEC CICS START command, or the trigger level on a transient data queue is reached. Often the started transaction is associated with a terminal, which may or may not be owned by the region in which the transaction runs.

Traditional routing of transactions started by ATI describes how CICS invokes the “terminal not known” global user exits, XICTENF and XALTENF, to deal with the situation where the terminal is not defined to the AOR.

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

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

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

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

   If the terminal is located on the first-choice TOR but not logged on, the TOR asks z/OS Communications Server for the applid of the generic resource member where the terminal is logged on. If the terminal is not logged on to any applid within the generic resource group, the ATI request is scheduled on the first-choice TOR. If the terminal is logged on to a different applid within the generic resource group, this information is passed to the AOR, and the ATI request is shipped to the correct TOR.

3. If the first-choice TOR is not available (and such an inquiry is possible) the AOR asks z/OS Communications Server for the location of the terminal. The inquiry is possible when all of the following are true:

   - The z/OS Communications Server in the AOR is version 4.2 or later (that is, it supports generic resources).
   - The AOR was started with the z/OS Communications Server system initialization parameter set to 'YES'.
   - The z/OS Communications Server generic resource name where the terminal may be logged on is known to the AOR. Such information is obtained from the skeleton TCTTE representing the remote terminal. If the first choice TOR name has been supplied by the user terminal-not-known exit, such an inquiry is not possible. Note that the inquiry will fail if the terminal is not logged on to the z/OS Communications Server generic resource name found in the skeleton TCTTE.

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

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

z/OS Communications Server knows the terminal by its netname, not by its CICS terminal identifier (TERMID). If there is a terminal definition in the AOR at the time the START is issued, CICS obtains the netname from that definition. If there is not, your terminal-not-known exit program should return:

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

Note:

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

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

Example 1:

1. A user logs on using the generic resource name CICS, which is the name of a set of TORs (TOR1 through TOR6). The user is connected to TOR1, because it is the most lightly loaded.
2. The user runs a transaction, which is routed to an AOR, AOR1. The terminal definition is shipped to AOR1.
3. The transaction issues an EXEC CICS START request, to start another transaction, after an interval, against the same terminal. The second transaction, like the first, is located on AOR1.
4. After the first transaction has completed, the user logs off; and logs on again later to collect the output from the second transaction. When logging on the second time, again using the generic resource name CICS, the user is connected to TOR2 because that is now the most lightly loaded.

5. The interval specified on the START request expires. However, the terminal is no longer defined to TOR1. The shipped terminal definition has not yet been deleted from AOR1 by the timeout delete mechanism.

- **Result:**

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

Example 2:

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

- **Result:**

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

  – The netname of the user's terminal

  – The name of a connection to any member of the generic resource that is likely to be currently active.

  CICS is then able to query z/OS Communications Server, as described in Example 1, and ship the request to the correct TOR (TOR2).

Using the ISSUE PASS command

The **`EXEC CICS ISSUE PASS`** command can be used to disconnect a terminal from CICS and transfer it to the z/OS Communications Server application specified on the LUNAME option.

For example, to transfer a terminal from this CICS to another terminal-owning region, you could issue the command:

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

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

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

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

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

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

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

Rules checklist

Here is a checklist of the rules that govern CICS use of the z/OS Communications Server generic resources function.

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

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

  Therefore, you are strongly recommended not to try to access such partners from more than one member of a generic resource.
- A system cannot statically define both an APPC generic resource name connection and an APPC member name connection to the same generic resource. (Generic resource name connections and member name connections are described in "Establishing connections between CICS TS for z/OS generic resources" on page 139.)

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

Dealing with special cases

This section describes some special cases that you may need to consider.

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

Non-autoinstalled terminals and connections

Because members of a generic resource should be functionally equivalent, it is not recommended that you should predefine terminals to specific members of a generic resource.

Important:

Use autoinstall instead, and allow the z/OS Communications Server to balance the TORs' workload dynamically. However, there may be times, for example, while you are migrating an existing TOR into a generic resource, when it is necessary to use static definitions.

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

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

A sample ISTEXCGR exit program is supplied with the z/OS Communications Server 4.2. For details, see z/OS Communications Server: SNA Programming.

Outbound LU6 connections

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

Using a “hub”

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

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

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

Because (unless the partner is also a CICS TS for z/OS generic resource) an attempt by a generic resource member to connect to an LU6 partner will succeed only if the partner knows the TOR by its generic resource name, it follows that the partner can accept a connection to only one member of the generic resource at a time. In a configuration in which more than one member of a generic resource must connect to a remote system, you can choose a region within the CICSplex to act as a **network hub**. This means that all generic resource members daisy-chain their requests for services from remote systems through the hub.

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

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

Figure 47 on page 152 shows a network hub that is not a member of the generic resource.

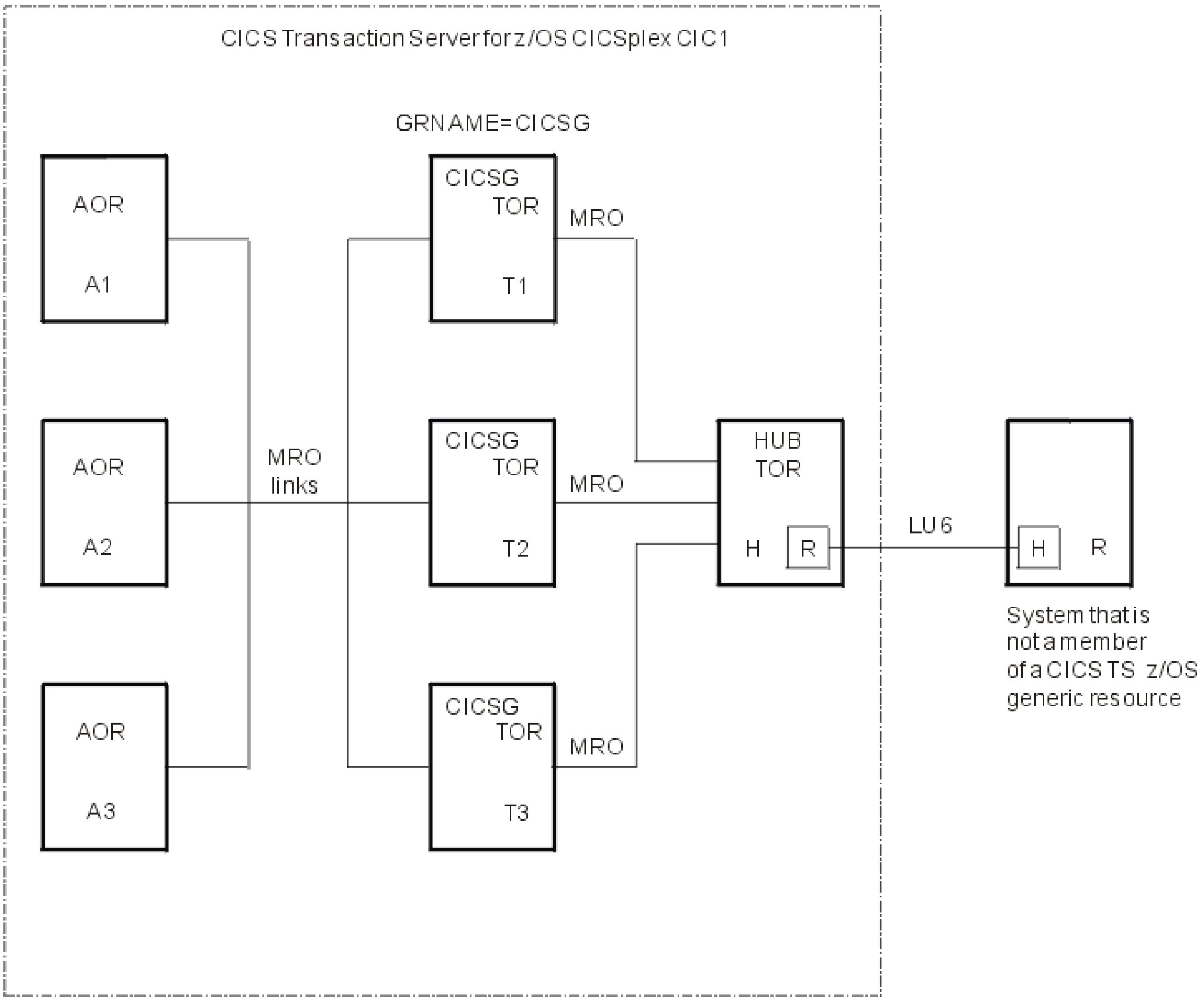

CICS Transaction Server for z/OS CICSplex CIC1
GRNAME=CICSG
AOR
A1
AOR
A2
AOR
A3
MRO
links
CICSG
TOR
T1
CICSG
TOR
T2
CICSG
TOR
T3
MRO
MRO
MRO
HUB
TOR
H
R
LU6
H
R
System that is not a member of a CICS TS z/OS generic resource

Figure 47. A network hub

In Figure 47 on page 152, the regions in CICSplex CIC1 are connected by MRO links. The terminal-owning regions T1, T2, and T3 are members of the generic resource group, CICSG, but the hub TOR, H, is not. H has an LU6.1 or APPC connection to the remote region, R. The TORs daisy-chain their requests to R through H.

Defining intercommunication resources

In an intercommunication environment, you create resources that define the links to other systems, and local definitions of remote resources.

The links to remote systems are:

- MRO links to other CICS regions
- MRO links for use by the external CICS interface
- IP interconnectivity (IPIC) links for use with distributed program link
- Multi-session APPC links to other APPC systems (CICS or non-CICS)
- Single-session APPC links to APPC terminals
- LUTYPE6.1 links to IMS systems.

Remote resources that can be defined to the local CICS system can be:

- Remote files

- Remote DL/I PSBs
- Remote transient-data queues
- Remote temporary-storage queues
- Remote terminals
- Remote APPC connections
- Remote programs
- Remote transactions

You might also need to define local resources required for ISC and MRO. These resources are obtained by including the relevant functional groups in the appropriate tables. However, you have the opportunity to modify some of the supplied definitions and to provide your own communication profiles.

Defining connections to remote systems

You can define and manage different types of connections between CICS regions or from CICS regions to non-CICS systems.

The types of connection that you can create are as follows:

- Connections for multiregion operation (MRO)
- Connections for use by the external CICS interface (EXCI)
- IPIC connections to remote regions
- ISC over SNA connections to remote systems, using logical unit type 6.2 (APPC) protocols
- ISC over SNA connections to remote IMS systems, using logical unit type 6.1 protocols
- Indirect connections for CICS transaction routing

Connections using the ACF/Communications Server application-to-application facilities are treated exactly as though they are intersystem connections and can be defined as either LUTYPE6.1 or APPC links.

How it works: connection definition in CICS

You can define different types of connections in CICS. You can use MRO and ISC over SNA (APPC and LUTYPE 6.1) connections or IP interconnectivity (IPIC) over TCP/IP connections.

MRO and ISC over SNA connections

The definition of an MRO or ISC over SNA connection to a remote system consists of two parts:

- The definition of the remote system itself
- The definition of sessions with the remote system

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

For single-session APPC terminals, you can use an alternative method of definition by using the TERMINAL and TYPETERM resources.

If the remote system is a CICS region or any other system that uses resource definition to define intersystem sessions, for example, IMS, the connection definition must match a compatible definition in the remote system. For remote systems with little or no flexibility in their session properties, for example, APPC terminals, the connection definition must match the fixed attributes of the remote system concerned.

IPIC connections

The definition of an IPIC connection between two CICS regions consists of two parts:

- The definition of the outbound attributes of the connection, including the target CICS region
- The definition of the inbound attributes of the connection, including the port number that CICS listens for requests

The local CICS region name

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

- Application identifier (APPLID)
- System identifier (SYSID)
- z/OS Communications Server generic resource name

All CICS regions have an APPLID and a SYSID. A terminal-owning region that is a member of a z/OS Communications Server generic resource group also has a z/OS Communications Server generic resource name. z/OS Communications Server generic resource names are described in "Configuring z/OS Communications Server generic resources" on page 133.

APPLID of the CICS region

The APPLID of a CICS system is the name by which it is known in the intercommunication network; that is, its netname.

- For MRO, CICS uses the APPLID name to identify itself when it signs on to the CICS interregion SVC, either during startup or in response to a **`SET IRC OPEN`** command.
- For ISC over SNA, the APPLID is used on a z/OS Communications Server APPL statement, to identify CICS to z/OS Communications Server.
- For IPIC, the APPLID attribute of an IPCONN resource identifies the APPLID of the remote system.

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

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

SYSID of the CICS region

The SYSID of a CICS region is a name (1–4 characters) known only to the CICS region itself. It is obtained (in order of priority) from:

1. The startup override
2. The SYSIDNT operand of the DFHSIT macro
3. The default value `CICS`.

The SYSID of your CICS region might also have to be specified in the DFHTCT TYPE=INITIAL macro if you are using macro-level resource definition. The only purpose of the SYSIDNT operand of DFHTCT TYPE=INITIAL is to control the assembly of local and remote terminal definitions in the terminal control table. The SYSID of a running CICS region is always the one specified by the system initialization parameters.

Identifying remote systems

In addition to having a SYSIDNT for itself, a CICS system requires a SYSIDNT for every other system with which it can communicate. SYSIDNT names are used to relate session definitions to system definitions; to identify the systems on which remote resources, such as files, reside; and to refer to specific systems in application programs.

SYSIDNT names are private to the CICS system in which they are defined; they are not known by other systems. In particular, the SYSIDNT defined for a remote CICS system is independent of the SYSIDNT by which the remote system knows itself; you need not make them the same.

The mapping between the local (private) SYSIDNT assigned to a remote system and the APPLID by which the remote system is known globally in the network (its netname), is made when you define the intercommunication link. For example, for an MRO or ISC over SNA connection, on the CONNECTION definition you specify the following attributes:

CONNECTION(*sysidnt*)
The local name for the remote system

NETNAME(*applid*)
The applid of the remote system

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

IPCONN(*sysidnt*)
The local name for the remote system

APPLID(*applid*)
The APPLID of the remote system

Each SYSIDNT name defined to a CICS system must be unique.

Defining IP interconnectivity (IPIC) connections

To define an IPIC connection, you create two resources, IPCONN and TCPIPSERVICE, on each CICS region that you want to connect. You can either create new IPIC connections , or you can migrate your existing APPC connections.

Before you begin

Restriction: IPIC supports specific intercommunication functions and releases. See the related links for this topic for more information.

TCP/IP services must be active in the CICS regions. You can activate TCP/IP services by setting the **`TCPIP`** system initialization parameter to YES.

Procedure

1. Define a TCPIPSERVICE resource to receive inbound requests on the local CICS region.

   The name of the TCPIPSERVICE resource must match the value of the TCPIPSERVICE attribute for the IPCONN resource.

   a) Specify the IP address on which this CICS region will listen in the HOST attribute. The ANY option specifies that CICS listens on any of the addresses known to TCP/IP for the host system.

      The host name can be up to 116 characters in length, or can be an IPv4 or IPv6 address. If you use an IPv6 address, ensure that you are operating in a dual-mode environment and that the client or server that you are communicating with is also operating in a dual-mode environment.

   b) Specify a port number on which the local CICS region listens for incoming client requests in the PORT attribute.

   c) Specify `IPIC` for the PROTOCOL attribute.

   d) Specify `NO` for the SOCKETCLOSE attribute.

   e) Specify the 4-character ID of the CICS transaction that runs the DFHISCOP program as the value of the TRANSACTION attribute.

      The default transaction for IPIC is CISS.

   f) Leave the SPECIFTCPS attribute blank.

    g) Optional: Specify the name of the IPCONN autoinstall user program as the value of the URM attribute.

       If you do not specify this attribute, CICS uses the CICS-supplied default IPCONN autoinstall user program, DFHISAIP. Specify NO to disable autoinstall.

2. Create an IPCONN resource on the local CICS region.
    a) Specify the IPCONN name. Specify a 4-character IPCONN name with four trailing spaces for CICS-to-CICS communications.
    b) Specify the host name in the HOST attribute, using the value that is specified in the TCPIPSERVICE resource in the remote CICS region.
       For example, `hostb.example.com`

       The host name can be up to 116 characters in length, or can be an IPv4 or IPv6 address. If you specify an IPv6 address (or a host name that resolves to an IPv6 address), ensure that you are operating in a dual-mode (IPv4 and IPv6) environment and that the client or server that you are communicating with is also operating in a dual-mode (IPv4 and IPv6) environment.
    c) Specify in the PORT attribute the port number on which the remote CICS region will listen.

       Specify `NO` if this IPCONN resource is not used for outbound requests and you are using the CICS Transaction Gateway.
    d) Specify the name of the TCPIPSERVICE resource on the local CICS region that specifies the inbound attributes of the IPIC connection as the value for the TCPIPSERVICE attribute.
    e) Specify values for the APPLID and NETWORKID attributes if you want to connect to a remote system that is in a different network.

       The combination of APPLID and NETWORKID attributes ensures that the remote CICS region is referred to by a unique name.
    f) Optional: Specify YES or `NO` for the INSERVICE attribute to set if you want the connection to be available when the resource is created.
    g) Specify values for the RECEIVECOUNT and SENDCOUNT attributes to set how many receive and send sessions are allowed for the IPIC connection.
3. Create a TCPIPSERVICE resource in the remote CICS region.
4. Create an IPCONN resource in the remote CICS region.

   Specify AUTOCONNECT(YES) to establish the connection between the two CICS regions.

Results

When the resources are enabled on the local and remote CICS regions, the connection is established between the CICS regions.

What to do next

You can use the IBM CICS Explorer or Web User Interface to view and update your IPIC connections. If you do not specify AUTOCONNECT(YES) for one of the IPCONN resources, you must acquire the connection by updating the status of the resource.

Defining IPIC high availability connections

To define an IPIC connection between a client region and an IPIC HA cluster, you need to create different sets of resources on the client regions to those on the cluster regions. Each client region requires a TCPIPSERVICE and an IPCONN. Each server region in the HA cluster requires two TCPIPSERVICEs and optionally one IPCONN, which can be predefined or auto-installed.

Figure 48 on page 157 shows the relationship between IPCONN and TCPIPSERVICE definitions in an IPIC HA environment.

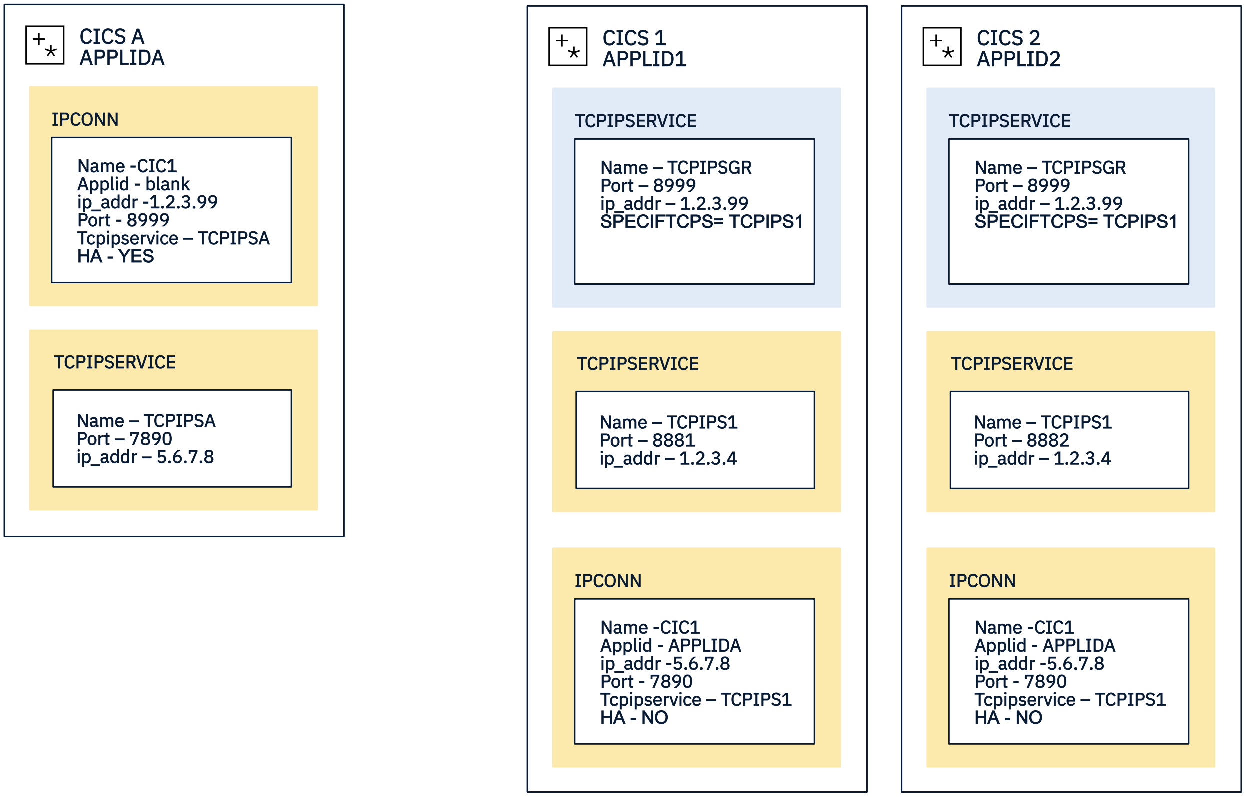


Figure 48. Related IPCONN and TCPIPSERVICE Definitions in an IPIC HA environment

CICS A is the client region, and CICS 1 and CICS 2 are regions in the HA cluster. The IPCONN installed in CICS A uses a port number which the generic TCPIPSERVICEs in both server regions are listening on. The generic TCPIPSERVICEs in both server regions have the name TCPIPSGR. The server regions both have specific TCPIPSERVICEs, each listening on its own unique port. When the client region connects through the shared port, that request is routed to one of the server regions. That server then returns to the client the port number from its specific TCPIPSERVICE. The client then establishes the connection using the new port, and all messages then go between the client and the selected server region for the lifetime of the connection.

For information on how to configure the network, see Connection balancing.

Configuring an HA server region
Each server region in the HA cluster requires two TCPIPSERVICEs and optionally one or more IPCONNs.

Before you begin

The specific TCPIPSERVICE resource must be brought into service before the generic TCPIPSERVICE, so you must define them in the order set out in this procedure.

Restriction: IPIC HA makes use of function provided in CICS TS V5.2 or later. Both client and cluster regions must be at this level. TCP/IP services must be active in the CICS regions. You can activate TCP/IP services by setting the **TCPIP** system initialization parameter to YES. IPIC HA connections must only be acquired by client regions. Attempts to acquire connections to an HA cluster can fail if there are other acquired IPIC connections between the client region and any regions within the cluster.

Procedure

1. Define a specific TCPIPSERVICE resource to receive inbound requests on the local CICS region.

   This resource must listen on an address and port that is exclusively reserved for this region.

   a) Specify the IP address on which this CICS region will listen in the HOST attribute.

The host name can be up to 116 characters in length, or can be an IPv4 or IPv6 address.

b) Specify a port number on which the local CICS region listens for incoming client requests in the PORT attribute.

c) Specify IPIC for the PROTOCOL attribute.

d) Specify NO for the SOCKETCLOSE attribute.

e) Specify the 4-character ID of the CICS transaction that runs the DFHISCOP program as the value of the TRANSACTION attribute.

The default transaction for IPIC is CISS.

f) Leave the SPECIFTCPS attribute blank.

g) Optional: Specify the name of the IPCONN autoinstall user program as the value of the URM attribute.

If you do not specify this attribute, CICS uses the CICS-supplied default IPCONN autoinstall user program, DFHISAIP. Specify NO to disable autoinstall.

2. Define a generic TCPIPSERVICE resource to receive inbound requests on the local CICS region. This resource must listen on an address and port that is shared by all the regions within the cluster.

   a) Specify the IP address on which this CICS region will listen in the HOST attribute.

   The host name can be up to 116 characters in length, or can be an IPv4 or IPv6 address.

   b) Specify a port number on which the local CICS region listens for incoming client requests in the PORT attribute.

   c) Specify IPIC for the PROTOCOL attribute.

   d) Specify NO for the SOCKETCLOSE attribute.

   e) Specify the 4-character ID of the CICS transaction that runs the DFHISCOP program as the value of the TRANSACTION attribute.

   The default transaction for IPIC is CISS.

   f) In addition set the SPECIFTCPS value to the name of the specific TCPIPSERVICE.

   g) If both resources are being installed from a single resource group, you are advised to choose a name for the generic resource that alphabetically follows the name of the corresponding specific TCPIPSERVICE.

   This is because CICS installs resources of the same type in their alphabetical order when they are stored in a common resource group. If, as a result of the alphabetical order of its name, the generic resource is installed before the specific one that it references, then the generic resource is left in an closed state and you have to take action to resolve this every time that you install the group.

3. Optionally define one or more IPCONN resources for the client regions that may connect to the HA cluster. If you do not define these then the specific TCPIPSERVICE must be configured to support IPCONN autoinstall processing.

   a) Specify the IPCONN name. Specify a 4-character IPCONN name with four trailing spaces for CICS-to-CICS communications.

   b) Specify the host name in the HOST attribute, using the value that is specified in the TCPIPSERVICE resource in the remote CICS client region. For example, hostb.example.com The host name can be up to 116 characters in length, or can be an IPv4 or IPv6 address.

   c) Specify in the PORT attribute the port number on which the remote CICS region will listen. This should be the same as the port specified in the TCPIPSERVICE resource in the client region.

   d) Specify the name of the specific TCPIPSERVICE resource on the local CICS region defined in step 1.

   e) Specify the application ID of the client region for the APPLID. If the client system is in a different network you must also specify the NETWORKID attribute.

   f) Specify NO for the HA attribute.

   g) Specify YES for the INSERVICE attribute to set if you want the connection to be available when the resource is created.

h) Specify NO for the AUTOCONNECT attribute, to prevent an attempt being made to acquire a connection to the client region when this resource is installed: HA connections **must** only be acquired from client regions.

i) Specify values for the RECEIVECOUNT and SENDCOUNT attributes to set how many receive and send sessions are allowed for the IPIC connection.

Results

When the resources are enabled on the local CICS regions, then that region will become part of the HA cluster and be able to process connection requests that client regions initiate.

Configuring an HA client region
Each client region requires a TCPIPSERVICE and an IPCONN.

Before you begin

Restriction: IPIC HA makes use of function provided in CICS TS V5.2 or later. Both client and cluster regions must be at this level. TCP/IP services must be active in the CICS regions. You can activate TCP/IP services by setting the **TCPIP** system initialization parameter to YES. When the client attempts to connect to the cluster, there must not be any acquired IPIC connections between a client region and any region in the HA cluster or failures can occur.

Procedure

1. Define a TCPIPSERVICE resource to receive inbound requests on the local CICS region.

   This resource must listen on an address and port that is exclusively reserved for this region.

   a) Specify the IP address on which this CICS region will listen in the HOST attribute. The ANY option specifies that CICS listens on any of the addresses known to TCP/IP for the host system.

      The host name can be up to 116 characters in length, or can be an IPv4 or IPv6 address.

   b) Specify a port number on which the local CICS region listens for incoming client requests in the PORT attribute.

   c) Specify IPIC for the PROTOCOL attribute.

   d) Specify NO for the SOCKETCLOSE attribute.

   e) Specify the 4-character ID of the CICS transaction that runs the DFHISCOP program as the value of the TRANSACTION attribute.

      The default transaction for IPIC is CISS.

   f) Leave the SPECIFTCPS attribute blank.

2. Define an IPCONN resource for the client regions to connect to the HA cluster.

   a) Specify the IPCONN name. Specify a 4-character IPCONN name with four trailing spaces if you are using a 3270 interface, in other cases the four spaces will be added automatically.

   b) Specify in the HOST attribute the host name of the HA cluster. This should resolve to the IP address specified in the generic TCPIPSERVICE used by the HA cluster regions.
      For example, `hostb.example.com`

      The host name can be up to 116 characters in length, or can be an IPv4 or IPv6 address.

   c) Specify in the PORT attribute the port number on which the HA cluster will listen. This should be the same as the port specified in the generic TCPIPSERVICE used by the HA cluster regions.

   d) Specify the name of the TCPIPSERVICE resource defined in step 1.

   e) Specify a unique value for the HA cluster as the APPLID. It must be unique from all other APPLIDs that have been used in other IPCONN resources installed in this region.

   f) Specify YES for the HA attribute, indicating that the APPLID refers to a cluster of regions.

   g) Optional: Specify YES or NO for the INSERVICE attribute to set if you want the connection to be available when the resource is created.

h) Specify values for the RECEIVECOUNT and SENDCOUNT attributes to set how many receive and send sessions are allowed for the IPIC connection.

Results

When the resources are enabled on the local CICS regions, then that region will be able to attempt to connect to the HA cluster.

What to do next

You can use the IBM CICS Explorer or Web User Interface to view and update your IPIC connections. If you do not specify AUTOCONNECT(YES) for the IPCONN resource, you must acquire the connection by updating the status of the resource.

Configuring an SSL connection for an IPIC high availability (HA) cluster
You need to configure the outbound SSL through the IPCONN that it uses to connect to the cluster.

The server must have the same set of SSL attributes in both the generic and specific TCPIPSERVICEs. If the SSL attributes are not the same the generic TCPIPSERVICE will be prevented from opening.

The connection is not intended for work to be routed from the server regions to client regions and it is not necessary to configure SSL for this purpose. The IPCONN resource in the server and the TCPIPSERVICE in the client region do not need to have the SSL attributes set.

Configuring IPIC connections for identity propagation

You define an IPCONN resource in a receiving CICS region to enable processing of incoming distributed identity information and you define an IPCONN resource in a sending region to specify whether a distributed identity is transmitted outside a sysplex.

Before you begin

You must configure your RACF RACMAP settings before you configure your IPIC connections, even if you have IDPROP(OPTIONAL) set in your IPCONN resource definition. Otherwise, you receive the RACF ICH408I message for every unmapped request that is sent to RACF.

About this task

Identity propagation over an IPIC connection relies on trusted connections between CICS regions or between CICS and CICS Transaction Gateway; for example, if CICS and CICS Transaction Gateway are not in the same sysplex, the connection must be over an SSL connection. Identity propagation over an IPIC connection needs a security manager that supports identity propagation. An ICRX identity token identifies the distributed identity of a user, and can be sent to CICS as part of a message.

If CICS receives an ICRX in a message that is sent over an IPIC connection, USERAUTH(IDENTIFY) must be defined for the IPCONN resource in the receiving CICS region to allow processing of the ICRX. If USERAUTH(IDENTIFY) is defined, CICS attempts to map the ICRX to an external security manager (ESM) user ID, for example, a RACF user ID. If the mapping is successful, the ESM user ID is used as the security context for the task that is attached to process the incoming message. If the ICRX cannot be mapped to an ESM user ID, because it is not defined to the external security manager, the message is processed as if it did not contain an ICRX. Local and remote START commands over an IPIC connection do not support identity propagation.

Procedure

1. Specify USERAUTH(IDENTIFY) in the IPCONN resource definition of the receiving CICS system.

   The IDENTIFY attribute specifies that incoming requests must include a user identifier, which can be provided in the form of an ICRX, but that client authentication is being managed by the security manager that is sending the request. If you are using CICS Transaction Gateway, you must specify USERAUTH(IDENTIFY) to allow CICS Transaction Gateway to pass the distributed identity to CICS. For more information about the IPCONN resource, see IPCONN resources.For more information about identity propagation with CICS Transaction Gateway, see the CICS Transaction Gateway information center.

2. Specify IDPROP(REQUIRED) in the IPCONN resource definition of the sending CICS system.

   The REQUIRED attribute specifies that a distributed identity is required for requests that use this connection, instead of a user ID. The attribute has no meaning if the connection is contained in a single sysplex or if either or both regions cannot support identity propagation. If the connection is between systems in the same sysplex, the connection operates as if IDPROP(OPTIONAL) is specified and ignores any other setting. The receiving CICS system must have USERAUTH(IDENTIFY) specified in the IPCONN resource to be able to process the distributed identity information.

Results

The distributed identity of a user can now be received in requests from a trusted security manager, for example, CICS Transaction Gateway, that are sent over an IPIC connection.

Migrating APPC and LUTYPE6.1 connections to IPIC

You can migrate your existing APPC and LUTYPE6.1 connections to IPIC connections. Existing connections continue to operate as before. The IPCONN definition takes precedence over the CONNECTION definition; that is, if an IPCONN and a CONNECTION have the same name, CICS uses the IPCONN definition.

Before you begin

If you want to migrate APPC and LUTYPE6.1 connections to IPIC, you must have installed support for IPIC. Activating IP interconnectivity (IPIC) connections describes how to do this.

About this task

The DFH0IPCC migration utility converts existing APPC and LUTYPE6.1 connections to IPIC. To migrate your existing connections to IPIC using the DFH0IPCC utility, complete the following steps.

Procedure

1. Create a TCPIPSERVICE resource definition in each of the interconnected regions.
   a) Specify PROTOCOL(IPIC).
   b) Specify TCPIPSERVICE(DFHIPIC) or TCPIPSERVICE(*servicename*).

      If you specify a user-defined name, use this same name for all the TCPIPSERVICE definitions that you create.
   c) Specify other options, such as PORTNUMBER, according to the requirements of the region where the TCPIPSERVICE definition is to be installed.
2. Put each TCPIPSERVICE definition in a resource definition group of its own.
3. Add one or more resource groups to each CICS system definition file (CSD) used by the interconnected regions, the number depending on the number of CICS regions the CSD serves and the number of unique TCPIPSERVICE definitions that they require.
4. Install one TCPIPSERVICE, named DFHIPIC, or user-defined service name, in each of the interconnected regions.
5. Complete an APPLID table for the interconnected CICS regions, as shown in Example 1.
   a) Create the table as a fixed-block, 80-byte record format.
   b) Fill the table using any method: manually, for example, or by a utility, such as a spreadsheet or script. You must preserve the fixed-length format.
      - You can remove or omit any of the provided comments or header lines in the table.
      - The table must contain the application identifiers (APPLIDs), network IDs, where applicable, TCP/IP port numbers, and host names of all the interconnected CICS regions.

- If the previously defined TCPIPSERVICE definitions were named anything other than DFHIPIC, the table must contain a .DEFAULT record with TCPIPSERVICE=*servicename* in the HOST column.

6. Copy your APPLID table to every system that contains a CSD used by the interconnected regions.
7. Create JCL that can be used to invoke DFH0IPCC through DFHCSDUP, like that shown in Example 2.

   Specify the lists and resource groups that you want DFH0IPCC to search for information about CONNECTION and SESSIONS definitions.

   The JCL issues a **DFHCSDUP EXTRACT** command, passing the utility program as the *USERPROGRAM*.
8. On one of the CSD-owning systems, use your customized JCL file to invoke the DFH0IPCC utility program.

   The utility program collects information about CONNECTION and SESSIONS definitions, creates IPCONN definitions, and writes a series of DEFINE statements, which form the SYSIN for your resulting DFHCSDUP invocation JCL.
9. Review the output produced by the utility program.
   a) Check that the IPCONN definitions are correct for your installation.

      You might want to modify the default SSL settings to add greater security controls for a particular connection.

   b) Modify the USER, PASSWORD, and library names in the generated JCL, to match those used by your location.
10. Run the generated JCL to add the new IPCONN resources to your CSD file.
11. Repeat steps 8, 9, and 10 for each CSD file used by the interconnected CICS regions.

Example

This example of an APPLID table shows the format that you must use. The table following the example has reference information for the table format.

```
********************************************************************************
*                                                                              *
* Description:                                                                 *
*     This Applid Table is for DFH0IPCC. This table must contain the           *
*     APPLIDs, NETWORKIDs (where applicable for foreign network connectivity), *
*     PORT numbers, and TCP/IP HOST names for all CICS regions in the systems  *
*     for which IPCONN definitions are to be created.                          *
*                                                                              *
* File Format:                                                                 *
*     This file must be in FB80 format, and relies on a tabular layout as      *
*     follows. Any characters can be used as separators. Add comments using an *
*     asterisk in the first column of the line. A HOST name that is too long   *
*     to fit into the table can be continued by placing an asterisk in column  *
*     80, and continuing on column 25 of the next row (the first column of the *
*     space for HOST). The APPLID field of any continuation record(s) must be  *
*     left blank.                                                              *
*                                                                              *
* Notes:                                                                       *
*     The optional .DEFAULT record (shown as follows) can be used to provide   *
*     either one or both of the following parameters:                          *
*     > A TCPIPSERVICE name, which must be provided immediately after          *
*       'TCPIPSERVICE=' in the HOST column. If a name is not provided, it      *
*       defaults to 'DFHIPIC'. In either case, this value is the name that must*
*       be used when defining the TCPIPSERVICEs for the CICS systems referred  *
*       to in this table.                                                      *
*     > A default NETWORKID, which must be provided in the NET-ID column.      *
*       Its omission results in the omission of the NETWORKID parameter in     *
*       the generated IPCONN definition statements for those APPLIDs that had  *
*       a blank NET-ID column.                                                 *
*                                                                              *
*     Examples of various valid table entries are shown following the .DEFAULT *
*     record. These are examples only. Ensure that all rows adhere to your     *
*     site's standards and conventions.                                        *
*                                                                              *
*     Important! When editing this file, ensure that the CAPS setting is OFF.  *
*     Otherwise, the case-sensitive HOST names might be destroyed.             *
*                                                                              *
********************************************************************************
*
********************************************************************************
APPLID. |NET-ID. |PORT.|HOST.
********************************************************************************
.DEFAULT|LOCALNET|     |TCPIPSERVICE=TCPSERV1
APPL1A  |        |9876 |my.local.hostname
OTHERCIC|OTHERNET|12345|this.host.has.a.very.long.name.which.is.going.to.requir*
        |        |     |e.a.continuation.record
* Comments such as this are entirely free-form other than the * in column 1
CICSXYZ |        |9875 |10.2.156.221
```

Figure 49. Example 1: APPLID table

Table 9. Format of APPLID table

Table column	Length	Description
APPLID	char 8	Unique identifier or .DEFAULT. The APPLID must match the NETNAME of the associated CONNECTION definition. See "Equivalent attributes on IPCONN definitions" on page 166. Use .DEFAULT to specify default values for NETID or TCPIPSERVICE. The leading dot prevents the word DEFAULT being used as a valid APPLID. Only one .DEFAULT row is allowed in the table.

Table 9. Format of APPLID table (continued)

Table column	Length	Description
Separator	char 1	Any alphanumeric character.
NETID	char 8	Network identifier. When left blank, the default NETID specified by the .DEFAULT row is used.
Separator	char 1	Any alphanumeric character.
PORT	char 5	Listening port number
Separator	char 1	Any alphanumeric character
HOST	char 55	TCP/IP host name
Continuation column	char 1	Normally blank. Any nonblank character in this field indicates that the host name is longer than 55 characters and continues in the HOST column in the following row.

You can use this example JCL to invoke DFH0IPCC through DFHCSDUP.

```
//IPCJOB   JOB user,CLASS=A,USER=user,PASSWORD=pass
/*ROUTE PRINT user
//CSDUPJOB EXEC PGM=DFHCSDUP,REGION=0M
//STEPLIB  DD DSN=loadlibrary,DISP=SHR
//         DD DSN=loadlibrary,DISP=SHR
//DFHCSD   DD DSN=csdfilename,DISP=SHR
//SYSPRINT DD SYSOUT=A
//CSDCOPY  DD UNIT=VIO
//APPLTABL DD DSN=applidtablename,
//            DISP=SHR,UNIT=SYSDA,SPACE=(CYL,(2,1)),
//            DCB=(RECFM=FB,BLKSIZE=15360,LRECL=80)
//LOGFILE     DD DSN=logfilename,
//            DISP=(MOD,CATLG,CATLG),UNIT=SYSDA,SPACE=(CYL,(2,1)),
//            DCB=(RECFM=FB,BLKSIZE=15360,LRECL=80)
//OUTFILE  DD DSN=outputfilename,
//            DISP=(MOD,CATLG,DELETE),UNIT=SYSDA,SPACE=(CYL,(2,1)),
//            DCB=(RECFM=FB,BLKSIZE=15360,LRECL=80)
//SYSUDUMP DD SYSOUT=A
//SYSABEND DD SYSOUT=A
//SYSIN    DD *
EXTRACT GR(group1) USERPROGRAM(DFH0IPCC) OBJECTS
EXTRACT GR(group2) USERPROGRAM(DFH0IPCC) OBJECTS
EXTRACT GR(list1) USERPROGRAM(DFH0IPCC) OBJECTS
EXTRACT GR(list2) USERPROGRAM(DFH0IPCC) OBJECTS
/*
//
```

Figure 50. Example 2: JCL to invoke DFH0IPCC through DFHCSDUP

The DFH0IPCC migration utility

The DFH0IPCC utility program that is provided with CICS converts existing APPC and LUTYPE6.1 connections to IPIC connections (IPCONNs). DFH0IPCC is a sample program for use with the CSD update batch utility program DFHCSDUP. The utility generates a set of statements that form the input to DFHCSDUP.

The DFH0IPCC program takes input supplied in a table that you can edit, called an *APPLID table*. This table is used to store the APPLIDs of all the regions in the relevant setup, with the corresponding HOST name of the region and the listening PORT of the TCPIPSERVICE definition used to deal with inbound TCP/IP connections.

The DFH0IPCC program examines lists and resource groups in the CSD for CICS regions, collecting information about the CONNECTION and SESSIONS definitions it finds. For each APPC or LUTYPE6.1 pair of CONNECTION and SESSIONS definitions, it creates an IPCONN definition. Where appropriate, the attributes of the IPCONN definition are taken from the CONNECTION and SESSIONS definitions, with the values of the remaining attributes taken from the APPLID table or allowed to take their default values. When the utility program has completed an IPCONN definition, it writes a series of DEFINE statements, which form the SYSIN for your resulting DFHCSDUP invocation JCL.

IPCONN attribute mapping

This table summarizes how the DFH0IPCC utility program maps the CONNECTION attributes to the IPCONN definition.

Table 10. IPCONN attribute mapping

IPCONN definition attribute	Migrated From or Created By	Comments
APPLID	CONNECTION (NETNAME)	Direct migration
AUTOCONNECT	CONNECTION (AUTOCONNECT)	Direct migration. But, if ALL, set the new value to YES.
CERTIFICATE	N/A	Blank
CIPHERS	N/A	Blank
DESCRIPTION	N/A	Blank. Not migrated. You can add this in the DFH0IPCC output.
GROUP	CONNECTION (GROUP) SESSIONS (GROUP)	Not changed
HOST	APPLID table	Must be specified in the APPLID table.
INSERVICE	CONNECTION (INSERVICE)	Direct migration
IPCONN	CONNECTION (CONNECTION)	Direct migration. See "IPCONN names" on page 166.
MAXQTIME	CONNECTION (MAXQTIME)	Direct migration
NETWORKID	APPLID table	No equivalent. Leave blank if not specified in the APPLID table or if using the default.
PORT	APPLID table	Must be specified in the APPLID table.
QUEUELIMIT	CONNECTION (QUEUELIMIT)	Direct migration
RECEIVECOUNT	Sum of SESSIONS (MAXIMUM)	Direct migration from the LUTYPE6.1 SESSIONS equivalent setting, or derived from the APPC SESSIONS MAXIMUM setting.
SECURITYNAME	CONNECTION (SECURITYNAME)	Direct migration from CONNECTION SECURITYNAME only.
SENDCOUNT	Sum of SESSIONS (MAXIMUM)	Direct migration from the LUTYPE6.1 SESSIONS equivalent setting, or derived from the APPC SESSIONS MAXIMUM setting.
SSL	N/A	Left blank. You can modify this in the DFH0IPCC output.

Table 10. IPCONN attribute mapping (continued)

IPCONN definition attribute	Migrated From or Created By	Comments
TCPIPSERVICE	APPLID table	Always “DFHIPIC” or as in the APPLID table. See “TCPIPSERVICE names” on page 166.
USERAUTH	CONNECTION (ATTACHSEC)	Direct migration from CONNECTION ATTACHSEC values LOCAL, IDENTIFY or VERIFY only.
XLNACTION	CONNECTION (XLNACTION)	Direct migration

IPCONN names

The IPCONN names are generated to avoid duplicates. The DFH0IPCC utility program uses the name of the CONNECTION definition because there is a one-to-one relationship between a CONNECTION definition and the IPCONN definition created from it. The coexistence of same-name CONNECTION and IPCONN definitions is fully supported by CICS provided that the CONNECTION NETNAME and IPCONN APPLID are the same. In this instance, CICS selects the IPCONN definition instead of the CONNECTION definition for routing of supported function.

TCPIPSERVICE names

Because an IPCONN definition cannot determine the TCPIPSERVICE name of a partner region, the utility cannot produce TCPIPSERVICE definitions; you must define them manually. The utility works in such a way that all TCPIPSERVICE names in regions for which the utility produces IPCONN definitions must be the same.

All IPCONN definitions created by the DFH0IPCC utility program have the default attribute, TCPIPSERVICE (DFHIPIC), unless you supply a different name using the .DEFAULT row in the APPLID file. If you specify another name, use that name for all TCPIPSERVICE definitions that you create.

Equivalent attributes on IPCONN definitions
If you want to migrate your APPC and LUTYPE6.1 connections manually, instead of running the DFH0IPCC migration utility, these tables show the attributes of CONNECTION and SESSION resource definitions for LUTYPE6.1 and APPC connections and the equivalent attributes on IPCONN definitions.

APPC connections

Table 11. Migrating APPC connections to IPIC. CONNECTION options and their IPCONN equivalents

CONNECTION options	APPC possible values	IPCONN equivalent value
ACCESSMETHOD	SNA	Not applicable
ATTACHSEC	LOCAL | IDENTIFY | VERIFY | PERSISTENT | MIXIDPE	USERAUTH LOCAL | IDENTIFY | VERIFY | NO | CERTIFICATE
AUTOCONNECT	NO | YES | ALL	NO | YES
BINDSECURITY	NO | YES	SSL NO | YES
DATASTREAM	USER	Not applicable
INDSYS	Not applicable (indirect connections only)	Not applicable (indirect connections only)
INSERVICE	YES | NO	As is
MAXQTIME	NO | 0 - 9999	As is

Table 11. Migrating APPC connections to IPIC. CONNECTION options and their IPCONN equivalents (continued)

CONNECTION options	**APPC possible values**	**IPCONN equivalent value**
NETNAME	The SNA APPLID of the remote region. (For XRF, the generic APPLID. For connections to an SNA generic resource, either the APPLID or generic resource name.)	Combination of APPLID and NETWORKID
PROTOCOL	APPC	Not applicable
PSRECOVERY	SYSDEFAULT | NONE	Not applicable
QUEUELIMIT	NO | 0 - 9999	As is
RECORDFORMAT	U	Not applicable
REMOTENAME	Name (sysid) by which the remote system is known to itself	Not applicable
REMOTESYSNET	APPLID of the remote system that owns the remote resource, if the link to the remote system is indirect	Not applicable
REMOTESYSTEM	Name (sysid) of the remote system, or sysid of the next system in the path, if the link to the remote system is indirect	Not applicable
SECURITYNAME	RACF ID of the remote system	As is
SINGLESESS	NO | YES	Not applicable
USEDFLTUSER	NO | YES	Not applicable
XLNACTION	KEEP | FORCE	As is

Table 12. Migrating APPC connections to IPIC. SESSIONS options and their IPCONN equivalents

SESSIONS options	**APPC possible values**	**IPCONN equivalent value**
AUTOCONNECT	NO | YES | ALL	Not applicable
BUILDCHAIN	YES	Not applicable
CONNECTION	Name of CONNECTION to which this SESSION definition applies to	Not applicable
DISCREQ	Not applicable	Not applicable
IOAREALEN	Not applicable	Not applicable
MAXIMUM	1 - 999, 0 - 999	SENDCOUNT & RECEIVECOUNT
MODENAME	Name of an SNA LOGMODE	Not applicable
NEPCLASS	Transaction class for the node error program	Not applicable
NETNAMEQ	Not applicable	Not applicable
PROTOCOL	APPC	Not applicable
RECEIVECOUNT	Not applicable	Derived from MAXIMUM
RECEIVEPFX	Not applicable	Not applicable

Table 12. Migrating APPC connections to IPIC. SESSIONS options and their IPCONN equivalents (continued)

SESSIONS options	APPC possible values	IPCONN equivalent value
RECEIVESIZE	RU size to receive: 1 - 30720	Not applicable
RECOVOPTION	SYSDEFAULT | CLEARCONV | RELEASESESS | UNCONDREL | NONE	Not applicable
RELREQ	NO | YES	Not applicable
SENDCOUNT	Not applicable	Derived from MAXIMUM
SENDPFX	Not applicable	Not applicable
SENDSIZE	RU size to send: 1 - 30720	Not applicable
SESSNAME	Not applicable	Not applicable
SESSPRIORITY	0 - 255	Not applicable
USERAREALEN	Length of TCTTE user area: 0 - 255	Not applicable
USERID	ID for sign on	Not applicable

LUTYPE6.1 connections

Table 13. Migrating LUTYPE6.1 connections to IPIC. CONNECTION options and their IPCONN equivalents

CONNECTION options	LUTYPE6.1 possible values	IPCONN equivalent value
ACCESSMETHOD	IRC | XM	Not applicable
ATTACHSEC	LOCAL | IDENTIFY	USERAUTH LOCAL | IDENTIFY | VERIFY | NO | CERTIFICATE
AUTOCONNECT	Not applicable	NO | YES
BINDSECURITY	Not applicable	SSL NO | YES
DATASTREAM	USER	Not applicable
INDSYS	Not applicable (indirect connections only)	Not applicable (indirect connections only)
INSERVICE	YES | NO	As is
MAXQTIME	NO | 0 - 9999	As is
NETNAME	The APPLID specified in the SIT of the remote region	`host.domain.country[:port]`
PROTOCOL	Blank	Not applicable
PSRECOVERY	Not applicable	Not applicable
QUEUELIMIT	NO | 0 - 9999	As is
RECORDFORMAT	U	Not applicable
REMOTENAME	Not applicable	Not applicable
REMOTESYSNET	Not applicable	Not applicable
REMOTESYSTEM	Not applicable	Not applicable
SECURITYNAME	Not applicable	As is

Table 13. Migrating LUTYPE6.1 connections to IPIC. CONNECTION options and their IPCONN equivalents (continued)		
CONNECTION options	**LUTYPE6.1 possible values**	**IPCONN equivalent value**
SINGLESESS	Not applicable	Not applicable
USEDFLTUSER	NO | YES	Not applicable
XLNACTION	KEEP | FORCE	As is

Table 14. Migrating LUTYPE6.1 connections to IPIC. SESSIONS options and their IPCONN equivalents		
SESSIONS options	**LUTYPE6.1 possible values**	**IPCONN equivalent value**
AUTOCONNECT	Not applicable	Not applicable
BUILDCHAIN	Not applicable	Not applicable
CONNECTION	Name of CONNECTION to which this SESSION definition applies	Not applicable
DISCREQ	Not applicable	Not applicable
IOAREALEN	Default TIOA size: 0 - 32767 , 0 - 32767	Not applicable
MAXIMUM	Not applicable	Not applicable
MODENAME	Not applicable	Not applicable
NEPCLASS	Transaction class for the node error program	Not applicable
NETNAMEQ	Not applicable	Not applicable
PROTOCOL	LU61	Not applicable
RECEIVECOUNT	Number of receive sessions: 1 - 999	As is
RECEIVEPFX	Termid prefix	Not applicable
RECEIVESIZE	Not applicable	Not applicable
RECOVOPTION	Not applicable	Not applicable
RELREQ	Not applicable	Not applicable
SENDCOUNT	Number of send sessions: 1 - 999	As is
SENDPFX	Termid prefix	Not applicable
SENDSIZE	Not applicable	Not applicable
SESSNAME	Not applicable	Not applicable
SESSPRIORITY	0 - 255	Not applicable
USERAREALEN	Length of TCTTE user area: 0 - 255	Not applicable
USERID	ID to sign in	Not applicable

Defining links for multiregion operation

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

Note: The external CICS interface (EXCI) uses a specialized form of MRO link, that is described in "Defining links for use by the external CICS interface" on page 172. This present section describes MRO links between CICS systems. However, most of its contents apply also to EXCI links, except where noted otherwise in "Defining links for use by the external CICS interface" on page 172.

From the point of view of the local CICS system, each session on the link is characterized as either a SEND session or a RECEIVE session. SEND sessions are used to carry an initial request from the local to the remote system and to carry any subsequent data flows associated with the initial request. Similarly, RECEIVE sessions are used to receive initial requests from the remote system.

Defining an MRO link

To define an MRO link, create a CONNECTION resource and an associated SESSIONS resource.

Procedure

1. Create a CONNECTION resource.

   Specify the following attributes:

   CONNECTION(*sysidnt*)
   sysidnt is the local name for the CICS system to which the link is being defined.

   NETNAME(*name*)
   The netname must be the name with which the remote system logs on to the interregion SVC; that is, its applid. If you do not specify a netname, then sysidnt must satisfy these requirements. There can be only one MRO link between any two CICS regions; that is, each CONNECTION must specify a unique netname.

   ACCESSMETHOD(IRC|XM)
   QUEUELIMIT(NO|0-9999)
   The maximum number of requests permitted to queue for free sessions to the remote system.

   MAXQTIME(NO|0-9999)
   The the maximum time between a queue becoming full and it being purged because the remote system is unresponsive. Further information is given in Intersystem session queue management.

   INSERVICE(YES)
   ATTACHSEC(LOCAL|IDENTIFY)
   USEDFLTUSER(NO|YES)

   For information about the ATTACHSEC and USEDFLTUSER security attributes see Specifying user security in link definitions.

   Do not specify a value for the PROTOCOL attribute - you specify the protocol in the SESSIONS resource.

2. Create a SESSIONS resource.

   If you are using RDO, the CONNECTION and SESSIONS must be in the same GROUP.

   Specify the following attributes:

   SESSIONS(*csdname*)
   CONNECTION(*sysidnt*)
   The CONNECTION attribute must match the sysidnt specified for the CONNECTION. Only one SESSIONS definition can be related to an MRO CONNECTION.

   PROTOCOL(LU61)

   RECEIVEPFX(*prefix1*) and SENDPFX(*prefix2*)

   Specify the prefixes which allow the sessions to be named. A prefix is a one-character or two-character string that is used to generate session identifiers (TRMIDNTs). If you do not specify prefixes, they default to '>' (for SEND) and '<' (for RECEIVE). It is recommended that you allow the prefixes to default, because:

- This guarantees that the session names generated by CICS are unique; prefixes must not cause a conflict with an existing connection or terminal name.
- If you specify your own 2-character prefixes, the number of sessions you can define for each connection is limited to 99. If you specify your own 1-character prefixes, the limit increases to 999—the same as for default prefixes—but you may find it harder to guarantee unique session names.

For an explanation of how CICS generates names for MRO sessions, see SESSIONS definition attributes

RECEIVECOUNT(*number1*)

SENDCOUNT(*number2*)

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

SESSPRIORITY(*number*) and IOAREALEN(*value*)

Choosing the access method for MRO

You can specify ACCESSMETHOD(XM) to select MVS cross-memory services for an MRO link. Cross-memory services are used only if the other end of the link also specifies cross-memory.

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

To select the CICS Type 3 SVC for interregion communication, use ACCESSMETHOD(IRC).

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

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

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

Example attributes for CONNECTION resource

CONNECTION(CICB)
The local name for remote system

NETNAME(CICSB)
The APPLID of remote system

ACCESSMETHOD(XM)
Use cross-memory services

QUEUELIMIT(NO)
If no sessions are free, queue all requests

INSERVICE(YES)

ATTACHSEC(LOCAL)
Use link security only

USEDFLTUSER(NO)

Example attributes for SESSIONS resource

SESSIONS(csdname)
Unique csd name

CONNECTION(CICB)
The name of the related CONNECTION resource

PROTOCOL(LU61)

RECEIVEPFX(<)

RECEIVECOUNT(5)
5 receive sessions

SENDPFX(>)

SENDCOUNT(3)
3 send sessions

SESSPRIORITY(100)

IOAREALEN(300)
Minimum TIOA size for sessions

Defining compatible MRO nodes

An MRO link must be defined in both of the systems that it connects. You must ensure that the two definitions are compatible with each other. For example, if one definition specifies six sending sessions, the other definition requires six receiving sessions.

About this task

The compatibility requirements are summarized in the following table. Related resources and attributes are shown by identical numbers.

Note: VTAM is now z/OS Communications Server.

	CICSA	**CICSB**
System initialization parameters	`APPLID=CICSA` 1	4 `APPLID=CICSB`

CONNECTION resource	`CONNECTION(CICB)` 2
	`NETNAME(CICSB)` 4
	`ACCESSMETHOD(IRC)`
	`QUEUELIMIT(500)`
	`MAXQTIME(500)`
	`INSERVICE(YES)`
		3 `CONNECTION(CICA)`
		1 `NETNAME(CICSA)`
		`ACCESSMETHOD(IRC)`
		`QUEUELIMIT(NO)`
		
		`INSERVICE(YES)`
		`ATTACHSEC(LOCAL)`

SESSIONS resource	`SESSIONS(csdname)`
	`CONNECTION(CICB)` 2
	`PROTOCOL(LU61)` 5
	`RECEIVEPFX(<)`
	`RECEIVECOUNT(8)` 6
	`SENDPFX(>)`
	`SENDCOUNT(10)` 7
		`SESSIONS(csdname)`
		3 `CONNECTION(CICA)`
		5 `PROTOCOL(LU61)`
		`RECEIVEPFX(<)`
		7 `RECEIVECOUNT(10)`
		`SENDPFX(>)`
		6 `SENDCOUNT(8)`

Note: VTAM is the previous name for z/OS Communications Server.

Defining links for use by the external CICS interface

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

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

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

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

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

Example CONNECTION attributes for a specific EXCI connection

CONNECTION(EIP1)
The local name for the connection

NETNAME(CLAP1)
The name of the user program specified on the EXCI INITIALIZE_USER command.

ACCESSMETHOD(IRC)

PROTOCOL(EXCI)

CONNTYPE(Specific)
Pipes are dedicated to a single user

INSERVICE(YES)

ATTACHSEC(LOCAL)

Example SESSIONS attributes for a specific EXCI connection

SESSIONS(csdname)
A unique csd name

CONNECTION(EIP1)
The name of the associated CONNECTION resource

PROTOCOL(EXCI)

RECEIVEPFX(<)

RECEIVECOUNT(5)
5 receive sessions

SENDPFX
Do not specify

SENDCOUNT
Do not specify

Example CONNECTION attributes for a generic EXCI connection

CONNECTION(EIP2)
The local name for the connection

ACCESSMETHOD(IRC)

NETNAME()
Must be blank for generic connection

INSERVICE(YES)

PROTOCOL(EXCI)

CONNTYPE(Generic)
Pipes are shared by multiple users

ATTACHSEC(LOCAL)

SESSIONS(csdname)
A unique csd name

CONNECTION(EIP2)
The name of the associated CONNECTION resource

PROTOCOL(EXCI)

RECEIVEPFX(<)

RECEIVECOUNT(5)
5 receive sessions

SENDPFX
Do not specify

SENDCOUNT
Do not specify

Figure 51. Example SESSIONS attributes for a generic EXCI connection

Installing MRO and EXCI link definitions

You can install *new* MRO and EXCI connections dynamically, while CICS is fully operational—there is no need to close down interregion communication (IRC) to do so.

Note that CICS commits the installation of connection definitions at the group level—if the installation of any connection or terminal fails, CICS backs out the installation of all connections in the group. Therefore, when adding new connections to a CICS region with IRC open, ensure that the new connections are in a group of their own.

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

For further information about installing MRO links, see CONNECTION attributes.

Defining APPC connections

An APPC connection consists of one or more sets of sessions. The sessions in each set have identical characteristics, apart from being either contention winners or contention losers.

Each set of sessions can be assigned a *modename* that enables it to be mapped to a z/OS Communications Server logmode name and from there to a class of service (COS). A set of APPC sessions is therefore referred to as a *modeset*.

An APPC terminal is often an APPC system that supports only a single session and which does not support an LU services manager. There are several ways of defining such terminals; further details are given under “Defining single-session APPC terminals” on page 178. This section describes the definition of one or more modesets containing more than one session.

To define an APPC connection to a remote system, you must create the following resources:

1. A CONNECTION resource to define the remote system.
2. A SESSIONS resource to define each set of sessions to the remote system.

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

For all APPC connections, except single-session connections to APPC terminals, CICS automatically builds a set of special sessions for the exclusive use of the LU services manager, using the modename SNASVCMG. This is a reserved name, and cannot be used for any of the sets that you define.

If you are defining a z/OS Communications Server logon mode table, remember to include an entry for the SNASVCMG sessions. See ACF/Communications Server LOGMODE table entries for CICS.

Defining the remote APPC system

A remote APPC system is defined with a CONNECTION resource.

To define a remote APPC system, create a CONNECTION resource with the following attributes:

NETNAME(name)

ACCESSMETHOD(VTAM)

> **Note:** VTAM is now z/OS Communications Server.

PROTOCOL(APPC)

SINGLESESS(NO)

QUEUELIMIT(NO|0-9999)

MAXQTIME(NO|0-9999)

AUTOCONNEC(NO|YES|ALL)

SECURITYNAME(value)

ATTACHSEC(LOCAL|IDENTIFY|VERIFY|PERSISTENT|MIXIDPE)

BINDPASSWORD(password)

BINDSECURITY(YES|NO)

USEDFLTUSER(NO|YES)

PSRECOVERY(SYSDEFAULT|NONE)

You must specify ACCESSMETHOD(VTAM) and PROTOCOL(APPC) to define an APPC system. The CONNECTION name (that is, the sysidnt) and the netname have the meanings explained in "Identifying remote systems" on page 154 (but see the box that follows).

Important:

If you are defining an APPC link to a terminal-owning region that is a member of a z/OS Communications Server generic resource group, NETNAME can specify either the TOR's *generic resource name*, or its applid. For advice on coding NETNAME for connections to a generic resource, see "Configuring z/OS Communications Server generic resources" on page 133.

Because this connection will have multiple sessions, you must specify SINGLESESS(N), or allow it to default. (The definition of single-session APPC terminals is described in "Defining single-session APPC terminals" on page 178.)

The AUTOCONNECT attribute specifies which of the sessions associated with the connection are to be bound when CICS is initialized. Further information is given in "The AUTOCONNECT attribute" on page 179.

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

If you are using z/OS Communications Server persistent session support, the PSRECOVERY attribute specifies whether sessions to the remote system are recovered, if the local CICS fails and restarts within the persistent session delay interval. Further information is given in "Using z/OS Communications Server persistent sessions on APPC links" on page 180.

For information about security options, see Implementing LU6.2 security.

Note: If the intersystem link is to be used by existing applications that were designed to run on LUTYPE6.1 links, you can use the DATASTREAM and RECORDFORMAT attributes to specify data stream information for asynchronous processing. The information provided by these attributes is not used by APPC application programs.

Defining groups of APPC sessions

Each group of sessions for an APPC system is defined by means of a SESSIONS resource.

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

Specify the following attributes:

SESSIONS(csdname)

CONNECTION(name)

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

MODENAME(name)

Specifies a name (1–8 characters) that identifies this group of related sessions. The name must be unique among the modenames for any one APPC intersystem link, and you must not use the reserved names SNASVCMG or CPSVCMG.

PROTOCOL(APPC)

MAXIMUM(m1,m2)

Specifies the maximum number of sessions that are to be supported for the group. The parameters of this option have the following meanings:

- **m1** specifies the maximum number of sessions in the group. The default value is 1.
- **m2** specifies the maximum number of sessions to be supported as contention winners. The number specified for m2 must not be greater than the number specified for m1. The default value for m2 is zero.

SENDSIZE(size)

The maximum size of request unit (RU) to be sent, in the range 256 - 30 720.

RECEIVESIZE(size)

The maximum size of request unit (RU) to be received, in the range 256 - 30 720.

SESSPRIORITY(number)

AUTOCONNECT(NO|YES|ALL)

Specifies whether the sessions are to be bound when CICS is initialized. Further information is given in "The AUTOCONNECT attribute" on page 179.

USERAREALEN(value)

RECOVOPTION(SYSDEFAULT|UNCONDREL|NONE)

If you are using z/OS Communications Server persistent session support, and CICS fails and restarts within the persistent session delay interval, the RECOVOPTION option specifies how CICS recovers the sessions. (The RECOVNOTIFY option does not apply to APPC sessions.) Further information is given in "Using z/OS Communications Server persistent sessions on APPC links" on page 180.

Defining compatible CICS APPC nodes

When you define an APPC link between two CICS systems, you must ensure that the definitions of the link in each of the systems are compatible.

The following table summarizes the compatibility requirements. Related options and operands are shown by identical numbers.

	CICSA	CICSB
System initialization parameters	`APPLID=CICSA` 1	3 `APPLID=CICSB`

	CICSA	CICSB

CONNECTION resource	CONNECTION(CICB) 2
	NETNAME(CICSB) 3
	ACCESSMETHOD(VTAM)
	PROTOCOL(APPC)
	SINGLESESS(N) 4
	QUEUELIMIT(500)
	MAXQTIME(500)
	BINDPASSWORD(pw) 5
		10 CONNECTION(CICA)
		1 NETNAME(CICSA)
		ACCESSMETHOD(VTAM)
		PROTOCOL(APPC)
		4 SINGLESESS(N)
		QUEUELIMIT(NO)
		ATTACHSEC(IDENTIFY)
		5 BINDPASSWORD(pw)

SESSIONS resource	SESSIONS(csdname) CONNECTION(CICB) 2 MODENAME(M1) 6 PROTOCOL(APPC) MAXIMUM(ss,ww) 7 SENDSIZE(kkk) 8 RECEIVESIZE(jjj) 9	SESSIONS(csdname) 10 CONNECTION(CICA) 6 MODENAME(M1) PROTOCOL(APPC) 7 MAXIMUM(ss,ww) 9 SENDSIZE(jjj) 8 RECEIVESIZE(kkk)

Notes:

VTAM is the previous name for z/OS Communications Server.

7 The values specified for MAXIMUM on either side of the link do not need to match, because they are negotiated by the LU services managers. However, a matching specification avoids unusable TCTTE entries, and also avoids unexpected bidding because of the "contention winners" negotiation.

8, 9 If the value specified for SENDSIZE on one side of the link does not match that specified for RECEIVESIZE on the other, CICS negotiates the values at BIND time.

Automatic installation of APPC links

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

Note: The method described here applies only to APPC parallel-session and single-session links initiated by BIND requests. The method to be used for APPC single-session links initiated by z/OS Communications Server CINIT requests is described in “Defining single-session APPC terminals” on page 178. You cannot autoinstall APPC parallel-session links initiated by CINIT requests.

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

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

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

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

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

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

For further information about using autoinstall with APPC links, see Autoinstalling APPC connections. For programming information about the autoinstall user program, see Writing a program to control autoinstall of APPC connections.

Defining single-session APPC terminals

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

Defining an APPC terminal – method 1

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

About this task

The CONNECTION and SESSIONS resources that are required are similar to those shown in "Defining the remote APPC system" on page 175 and "Defining groups of APPC sessions" on page 176. The differences are as follows:

- In the CONNECTION resource, you must specify SINGLESESS(Y)
- In the SESSIONS resource, you must specify MAXIMUM(1,0). The second value (0) has no meaning for a single session definition because CICS always binds as a contention winner. However, CICS accepts a negotiated bind, or a negotiated bind response, in which it is changed to the contention loser.

Defining an APPC terminal – method 2

You can define a single-session APPC terminal as a TERMINAL with an associated TYPETERM.

About this task

This method of definition has two principal advantages:

1. You can use a single TYPETERM for all your APPC terminals of the same type.
2. It makes the AUTOINSTALL facility available for APPC single-session terminals.

   Autoinstall for APPC single sessions initiated by a z/OS Communications Server VTAM CINIT works in the same way as autoinstall for other terminals, in that you must supply a TERMINAL—TYPETERM model pair. For further information about using autoinstall with APPC single-session terminals, see Autoinstalling APPC connections.

Because all APPC devices are seen as systems by CICS, the value that you define in the TERMINAL attribute is effectively a system name. When you inquire about an APPC terminal, you actually inquire about a CONNECTION.

A single, contention-winning session is implied when you create TERMINAL resource. However, for APPC terminals, CICS accepts a negotiated bind in which it is changed to the contention loser.

If you plan to use automatic installation for your APPC terminals, you need the model terminal definition (LU62) that is provided in the CICS-supplied CSD group DFHTERM. You also have to write an autoinstall user program, and provide suitable z/OS Communications Server LOGMODE entries.

Procedure

1. Create a TERMINAL resource with the following attributes:.

   TERMINAL(sysid)

   MODENAME(modename)

   TYPETERM(typeterm)

   Specify any other appropriate attributes
2. Create a TYPETERM resource with the following attributes:

TYPETERM(typeterm)

DEVICE(APPC)

Specify any other appropriate attributes. The CICS-supplied group DFHTYPE contains a TYPETERM, DFHLU62T, that is suitable for APPC terminals. You can either use this TYPETERM resource, or use it as the basis for your own definition.

The AUTOCONNECT attribute

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

Except for single-session APPC terminals (see "Defining single-session APPC terminals" on page 178), two events are necessary to establish sessions to a remote APPC system.

1. The connection to the remote system must be established. This means binding the LU services manager sessions (SNASVCMG) and carrying out initial negotiations.
2. The sessions of the modeset in question must be bound.

These events are controlled in part by the AUTOCONNECT attribute of the CONNECTION resource, and in part by the AUTOCONNECT attribute of the SESSIONS resource.

The AUTOCONNECT attribute of a CONNECTION resource

The AUTOCONNECT option specifies whether CICS is to try to bind the LU services manager sessions at the earliest opportunity (when the z/OS Communications Server ACB is opened).

It can have the following values:

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

AUTOCONNECT(YES)
specifies that CICS **is** to try to bind the LU services manager sessions.

AUTOCONNECT(ALL)
the same as YES.

The LU services manager sessions cannot, of course, be bound if the remote system is not available. If for any reason they are not bound during CICS initialization, they can be bound when the connection is placed into INSERVICE ACQUIRED state. They are also bound if the remote system itself initiates communication. For a single-session APPC terminal, the AUTOCONNECT attribute has no effect. This is because a single-session connection has no LU services manager.

The AUTOCONNECT attribute of the SESSIONS resource

The AUTOCONNECT attribute specifies which sessions are to be bound when the associated LU services manager sessions have been bound. No user sessions can be bound before this time.

The option can have the following values:

AUTOCONNECT(NO)
specifies that no sessions are to be bound.

AUTOCONNECT(YES)
specifies that the contention-winning sessions are to be bound.

AUTOCONNECT(ALL)
specifies that the contention-winning and the contention-losing sessions are to be bound.

AUTOCONNECT(ALL) allows CICS to bind contention-losing sessions with remote systems that cannot send bind requests. By specifying AUTOCONNECT(ALL), you can cause CICS to bind a number of contention winners other than the number originally specified in the local system. The number of contention winners that CICS binds depends on the reply that the partner system gives to the request to initiate sessions (CNOS exchange). CICS tries to bind as contention winners all sessions that are not designated as contention losers in the CNOS reply.

For example, suppose that you specify MAXIMUM(10,4) in the local system and MAXIMUM(10,2) in the remote system. If the sessions are acquired from the local system, and the contention-losing sessions bind successfully, the result is 8 primary contention-winning sessions.

Important: Never specify AUTOCONNECT(ALL) for sessions to another CICS system, or to any system that can send a bind request. This could lead to bind-race conditions that CICS cannot resolve.

If AUTOCONNECT(NO) is specified, the sessions can be bound and made available by setting the modename into ACQUIRED AVAILABLE command. If this is not done, sessions are bound individually according to the demands of your application program.

For a single-session APPC terminal, the value specified for the AUTOCONNECT attribute of the SESSIONS or TYPETERM resources determines whether CICS tries to bind the single session or not.

Using z/OS Communications Server persistent sessions on APPC links

You can use z/OS Communications Server persistent sessions to improve the availability of APPC links. z/OS Communications Server persistent sessions support enables sessions to be recovered without the need for network flows in the event of a CICS or z/OS Communications Server failure.

Recovery with z/OS Communications Server persistent sessions explains what happens when you use persistent sessions support, and why you might want to run a CICS region without persistent sessions support.

If APPC sessions are active at the time of the CICS, Communications Server or z/OS failure, persistent sessions recovery appears to APPC partners as CICS hanging. The Communications Server saves requests issued by the APPC partner, and passes them to CICS when recovery is complete. When CICS reestablishes a connection with the Communications Server, recovery of terminal sessions is determined by the settings for the PSRECOVERY option of the CONNECTION resource definition and the RECOVOPTION option of the SESSIONS resource definition. You must set the PSRECOVERY option of the CONNECTION resource definition to the default value SYSDEFAULT for sessions to be recovered. The alternative, NONE, means that no sessions are recovered. If you have selected the appropriate recovery options and the APPC sessions are in the correct state, CICS performs an **ISSUE ABEND** to inform the partner that the current conversation has been abnormally ended.

The PSRECOVERY attribute of the CONNECTION resource

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

SYSDEFAULT

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

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

NONE

All sessions are unbound as out-of-service with no CNOS recovery.

The RECOVOPTION attribute of SESSIONS and TYPETERM resources

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

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

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

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

Defining logical unit type 6.1 links

LUTYPE6.1 links are necessary for intersystem communication between CICS and any system, such as IMS, that supports LUTYPE6.1 protocols but does not fully support APPC. You are advised to use MRO or APPC links for CICS-to-CICS communication.

Restriction:

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

A CONNECTION resource is always required to define the remote system on an LUTYPE6.1 link. The sessions, however, can be defined in either of the following ways:

1. By using a single SESSIONS resource to define a pool of sessions with identical characteristics.
2. By using a separate SESSIONS resource to define each individual session. This method must be used to define sessions with systems, such as IMS, that require individual sessions to be explicitly named.

Defining CICS-to-IMS LUTYPE6.1 links

A link to an IMS system requires a definition of the connection (or system) and a separate definition of each of the sessions.

Create a CONNECTION resource with the following attributes:

CONNECTION(sysidnt)

NETNAME(name)

ACCESSMETHOD(VTAM)

Note: VTAM is the previous name for z/OS Communications Server.

PROTOCOL(LU61)

DATASTREAM(USER|3270|SCS|STRFIELD|LMS)

RECORDFORMAT(U|VB)

QUEUELIMIT(NO|0-9999)

MAXQTIME(NO|0-9999)

INSERVICE(YES)

SECURITYNAME(name)

ATTACHSEC(LOCAL)

For each session, create a SESSIONS resource with the following attributes:

SESSIONS(csdname)

CONNECTION(sysidnt)

SESSNAME(name)

NETNAMEQ(name)

PROTOCOL(LU61)

RECEIVECOUNT(1|0)

SENDCOUNT(0|1)

SENDSIZE(size)

RECEIVESIZE(size)

SESSPRIORITY(number)

AUTOCONNECT(NO|YES|ALL)

BUILDCHAIN(YES)

IOAREALEN(value)

Defining compatible CICS and IMS nodes

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

An overview of IMS system definition is given in "Configuring support for ISC over SNA" on page 132. The relationships between CICS and IMS definitions are summarized in "Other session parameters" on page 183.

System names
The network name of the CICS system (its applid) is specified on the APPLID CICS system initialization parameter.

This name must be specified on the NAME operand of the IMS TERMINAL macro that defines the CICS system. For CICS systems that use XRF, the name will be the CICS generic applid. For non-XRF CICS systems, the name will be the single applid specified on the APPLID system initialization parameter.

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

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

You must specify the network name of the IMS system in the NETNAME attribute of the CONNECTION resource that defines the IMS system.

Number of sessions
In IMS, the number of parallel sessions that are required between the CICS and IMS system must be specified in the SESSION operand of the IMS TERMINAL macro.

Each session is then represented by a SUBPOOL entry in the IMS VTAMPOOL. In CICS, each of these sessions is represented by an individual session definition.

Session names
Each CICS-to-IMS session is uniquely identified by a session-qualifier pair, which is formed from the CICS name for the session and the IMS name for the session.

The CICS name for the session is specified in the SESSNAME attribute of the SESSIONS resource. For sessions that are to be initiated by IMS, this name must correspond to the ID parameter of the IMS OPNDST command for the session. For sessions initiated by CICS, the name is supplied on the CICS OPNDST command and is saved by IMS.

The IMS name for the session is specified in the NAME operand of the IMS SUBPOOL macro. You must make the relationship between the session names explicit by coding this name in the NETNAMEQ attribute of the corresponding SESSIONS resource.

The CICS and the IMS names for a session can be the same, and this approach is recommended for operational convenience.

Other session parameters
This topic lists the remaining attributes of the CONNECTION and SESSIONS resources that are of significance for CICS-to-IMS sessions.

ATTACHSEC
Must be specified as LOCAL.

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

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

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

This option is used only when CICS is communicating with IMS by using the START command (asynchronous processing). CICS messages generated by the START command always cause IMS to interpret the data stream profile as input for component 1.

The data stream profile for distributed transaction processing can be specified by the application program by means of the DATASTR option of the BUILD ATTACH command.

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

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

RECORDFORMAT(U|VB)
Specifies the type of chaining that CICS is to use for transmissions on this session that are initiated by START commands (asynchronous processing).

Two types of data-handling algorithms are supported between CICS and IMS:

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

Variable-length variable-blocked records (VLVB)
Messages are sent in variable-length variable-blocked format with a halfword length field before each record. This format corresponds to RECORDFORMAT(VB).

The data stream format for distributed transaction processing can be specified by the application program by means of the RECFM option of the BUILD ATTACH command.

Additional information on these data formats is given in CICS-to-IMS applications.

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

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

- `SENDCOUNT(1)`
- Allow `RECEIVECOUNT` to default. Do *not* specify `RECEIVECOUNT(0)`.

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

- `RECEIVECOUNT(1)`
- Allow `SENDCOUNT` to default. Do *not* specify `SENDCOUNT(0)`.

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

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

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

SENDSIZE and RECEIVESIZE

Specify the maximum z/OS Communications Server request unit (RU) sizes for these sessions.

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

The compatibility requirements are summarized in the following table. Related options and operands are shown by identical numbers.

Note: VTAM is now z/OS Communications Server.

CICS	**IMS**

System initialization parameters	`COMM APPLID=SYSIMS`
	7 `RECANY=nnn+22`
	`EDTNAME=ISCEDT`
	
	4 `TYPE UNITYPE=LUTYPE6`
	
	1 `TERMINAL NAME=SYSCICS`
	`SESSION=2`
	`COMPT1`
	`COMPT2`
	6 `OUTBUF=mmm`
	
	`VTAMPOOL`
	
	5 `SUBPOOL NAME=CIC1`
	
	`NAME CICLT1 COMPT=1`
	
	`NAME CICLT1A`
	
	8 `SUBPOOL NAME=CIC2`
	
	`NAME CICLT2 COMPT=2`
	
	3 `DFSHSBxx USERVAR=SYSIMS`

`APPLID=SYSCICS` 1	
CONNECTION resource	

`CONNECTION(IMSR)` 2
`NETNAME(SYSIMS)` 3
`ACCESSMETHOD(VTAM)`
`PROTOCOL(LU61)`
`DATASTREAM(USER)`
`ATTACHSEC(LOCAL)`	

SESSIONS resources	

`SESSIONS(csdname1)`
`CONNECTION(IMSR)` 2
`SESSNAME(IMS1)`
`NETNAMEQ(CIC1)` 5
`PROTOCOL(LU61)` 4
`SENDCOUNT(1)`
`SENDSIZE(nnn)` 7
`RECEIVESIZE(mmm)` 6
`IOAREALEN(nnn,16364)`
 	
`SESSIONS(csdname1)`
`CONNECTION(IMSR)` 2
`SESSNAME(IMS2)`
`NETNAMEQ(CIC2)` 8
`PROTOCOL(LU61)` 4
`SENDCOUNT(1)`
`SENDSIZE(nnn)` 7
`RECEIVESIZE(mmm)` 6
`IOAREALEN(nnn,16364)`	

Note: For an example of a z/OS Communications Server logmode table entry for IMS, see ACF/SNA LOGMODE table entries for IMS .

Defining multiple links to an IMS system

You can define more than one intersystem link between a CICS and an IMS system.

About this task

To defined multiple links to an IMS system, create two or more CONNECTION definitions (with their associated SESSION definitions) with the same netname, but with different sysidnts. Although all the system definitions resolve to the same netname, and therefore to the same IMS system, the use of a sysidnt name in CICS causes CICS to allocate a session from the link with the specified sysidnt.

It is recommended that you define up to three links (that is, groups of sessions) between a CICS and an IMS system, depending upon the application requirements of your installation:

1. For CICS-initiated distributed transaction processing (synchronous processing).

   CICS applications that use the SEND/RECEIVE interface can use the sysidnt of this group to allocate a session to the remote system. The session is held (busy) until the conversation is terminated.
2. For CICS-initiated asynchronous processing.

   CICS applications that use the START command can name the sysidnt of this group. CICS uses the first non-busy session to ship the start request.

   IMS sends a positive response to CICS as soon as it has queued the start request, so that the session is in use for a relatively short period. Consequently, the first session in the group shows the heaviest usage, and the frequency of usage decreases towards the last session in the group.
3. For IMS-initiated asynchronous processing.

   This group is also useful as part of the solution to a performance problem that can arise with CICS-initiated asynchronous processing. An IMS transaction that is initiated as a result of a START command shipped on a particular session uses the same session to ship its "reply" START command to CICS. For the reasons given previously in (2), the CICS START command was probably shipped on the busiest session and, because the session is busy and CICS is the contention winner, the replies from IMS may be queuing for a chance to use the session.

   However, facilities exist in IMS for a transaction to alter its default output session, and a switch to a session in this third group can reduce this sort of queuing problem.

Table 15. Defining multiple links to an IMS node

CICS
System Initialization parameters:

`SYSIDNT=CICL,`
`APPLID=SYSCICS`

Resources for CICS-initiated distributed transaction processing

`CONNECTION(IMSA)`
`NETNAME(SYSIMS)`
`ACCESSMETHOD(VTAM)`
 	
`SESSIONS(csdname)`
`CONNECTION(IMSA)`
`SESSNAME(IMS1)`
`NETNAMEQ(DTP1)`
`PROTOCOL(LU61)`
 	
`SESSIONS(csdname)`
`  .`
`  .`

Resources for CICS-initiated asynchronous processing

Table 15. Defining multiple links to an IMS node (continued)

CICS

```
CONNECTION(IMSB)
NETNAME(SYSIMS)
ACCESSMETHOD(VTAM)

SESSIONS(csdname)
CONNECTION(IMSB)
SESSNAME(IMS1)
NETNAMEQ(ASP1)
PROTOCOL(LU61)

SESSIONS(csdname)
  .
  .
```

Resources for IMS-initiated asynchronous processing

```
CONNECTION(IMSC)
NETNAME(SYSIMS)
ACCESSMETHOD(VTAM)

SESSIONS(csdname)
CONNECTION(IMSC)
SESSNAME(IMS1)
NETNAMEQ(IST1)
PROTOCOL(LU61)

SESSIONS(csdname)
  .
  .
```

Note: VTAM is the previous name for z/OS Communications Server.

Defining indirect links for transaction routing

In some older releases of CICS (no longer supported), indirect links between CICS regions were required for transaction routing across intermediate regions. In a network consisting solely of currently-available CICS systems, indirect links are only required if you are using non-z/OS Communications Server terminals. Optionally, you can define them for use with z/OS Communications Server terminals. Indirect links are never used for function shipping, distributed program link, asynchronous processing, or distributed transaction processing.

The following figure shows the concept of an indirect link.

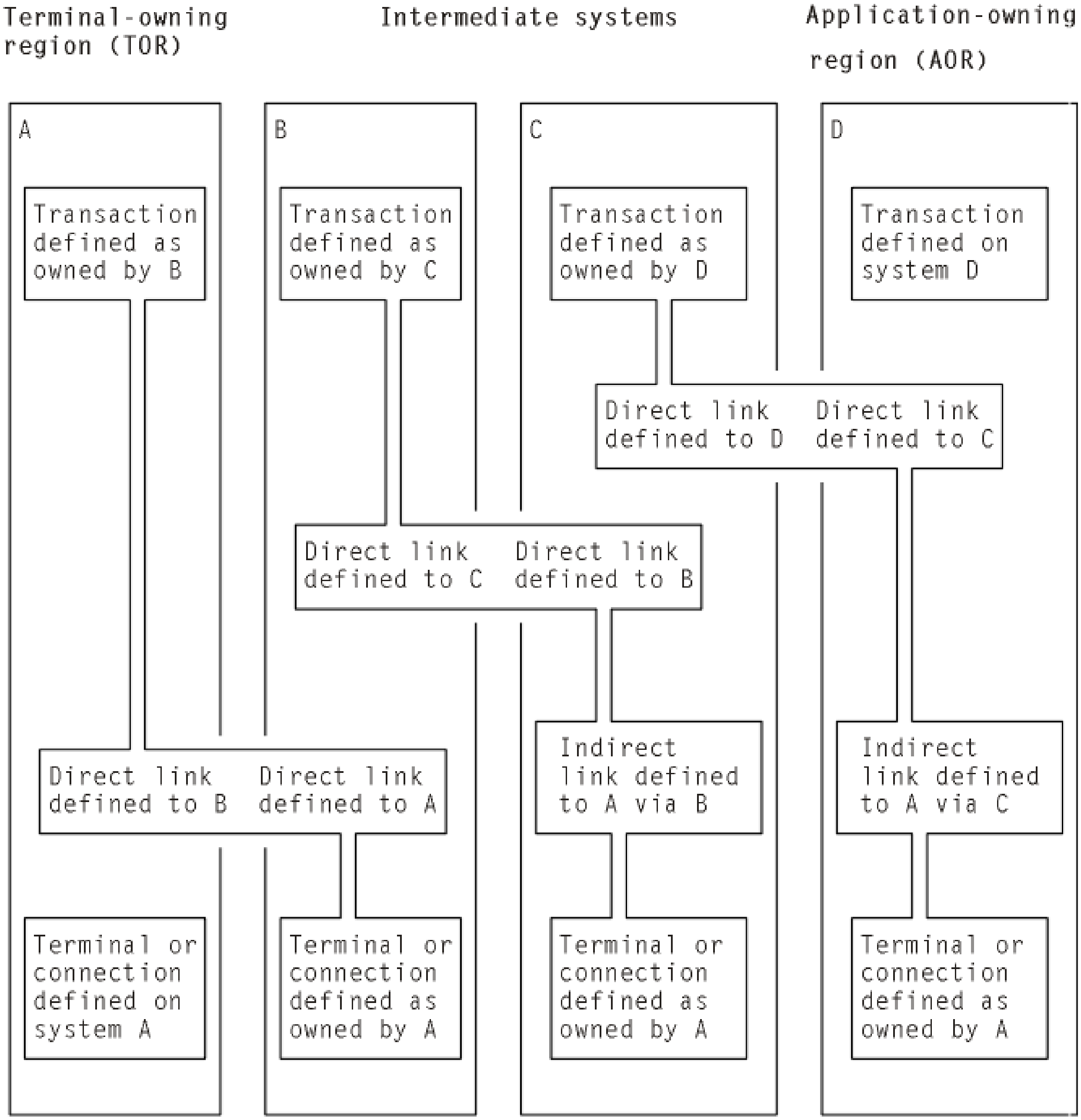

Terminal-owning region (TOR)
Intermediate systems
Application-owning region (AOR)
A
B
C
D
Transaction defined as owned by B
Transaction defined as owned by C
Transaction defined as owned by D
Transaction defined on system D
Direct link defined to D
Direct link defined to C
Direct link defined to C
Direct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

Figure 52. Indirect links for transaction routing

This figure illustrates a chain of systems (A, B, C, D) linked by MRO or APPC links (you cannot do transaction routing over LUTYPE6.1 links).

It is assumed that you want to establish a transaction-routing path between a terminal-owning region A and an application-owning region D. There is no direct link available between system A and system D, but a path is available via the **intermediate** systems B and C.

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

Defining indirect links in CICS Transaction Server for z/OS

CICS systems reference remote terminals using a unique identifier that is formed from the applid (netname) of the terminal-owning region (TOR) and the identifier by which the terminal is known on the terminal-owning region.

For more information on remote resource definition, see "Defining remote resources" on page 200.

CICS must have access to the netname of the TOR to be able to form the fully-qualified terminal identifier. In old releases of CICS (no longer supported), an indirect link definition had two purposes. Where there was no direct link to the TOR, it:

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

Thus, in Figure 52 on page 187, the indirect link definition in system D provides the netname of system A and identifies system C as the next system in the path. Similarly, the indirect link definition in system C provides the netname of system A and identifies system B as the next system in the path. System B has a direct link to system A, and therefore does not require an indirect link.

In CICS Transaction Server for z/OS, unless you are using non-z/OS Communications Server terminals, indirect links are optional. Different considerations apply, depending on whether you are using shippable or hard-coded terminal definitions.

Shippable terminals

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

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

Note: Non-z/OS Communications Server terminals are not shippable.

Hard-coded terminals

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

If you are using non-z/OS Communications Server terminals, indirect links are required. This is because must use the DFHTCT TYPE=REMOTE or TYPE=REGION macros to define non-z/OS Communications Server terminals, and these do not include an equivalent of the REMOTESYSNET attribute.

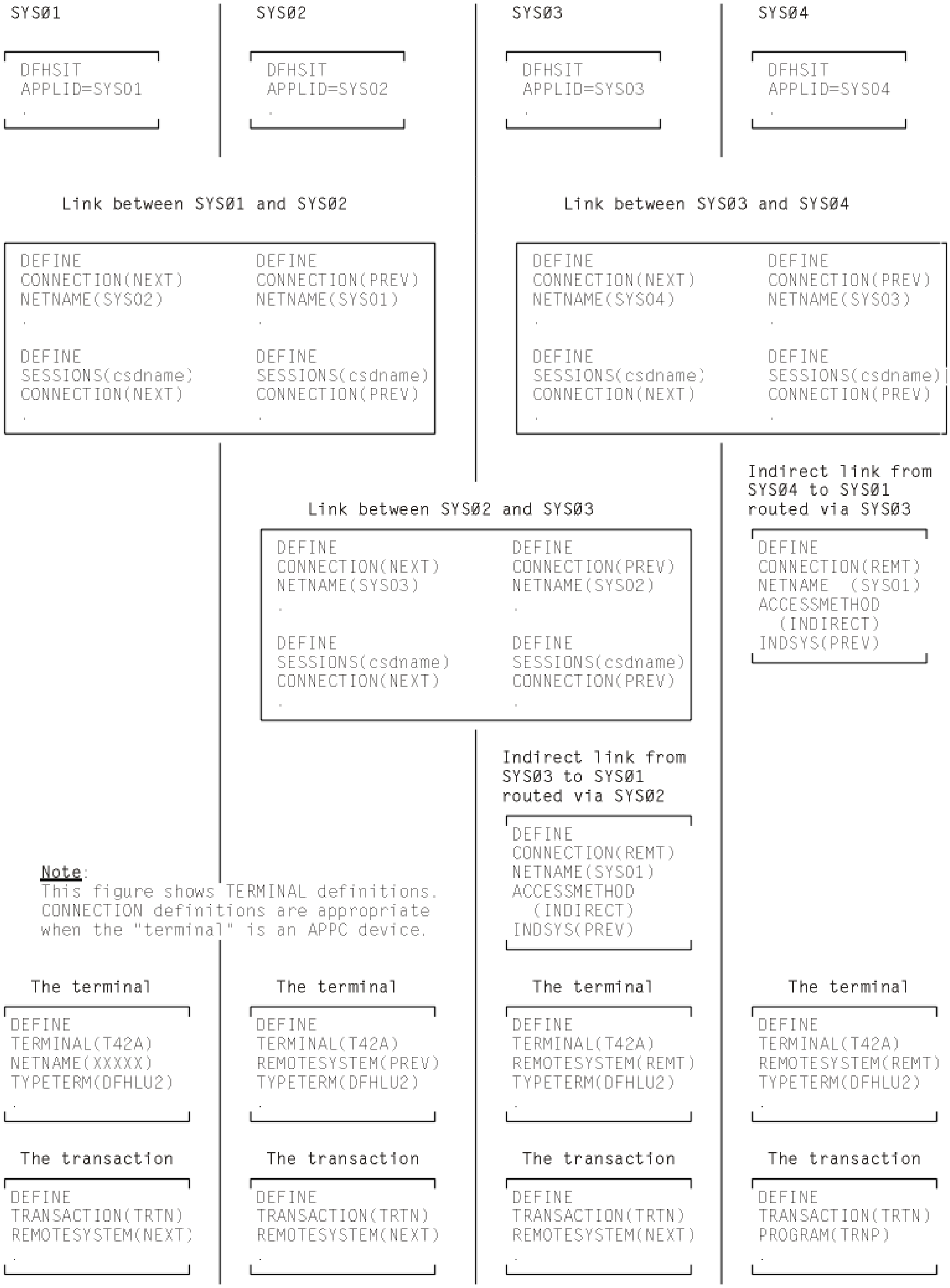
Therefore, in CICS Transaction Server for z/OS, you might decide to define indirect links:

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

Resource definition for transaction routing using indirect links

The resource definitions that are required to establish a transaction-routing path between a terminal-owning region SYS01 and an application-owning region SYS04 via two intermediate systems SYS02 and SYS03, using indirect links, are shown.

The resource definitions required are shown in Figure 53 on page 190. For clarity, the figure shows hard-coded remote terminal definitions that do not use the REMOTESYSNET option (if REMOTESYSNET is used, indirect links are not required). Shippable terminals could also be used.



SYS01

```
DFHSIT
APPLID=SYS01
.
```

SYS02

```
DFHSIT
APPLID=SYS02
.
```

SYS03

```
DFHSIT
APPLID=SYS03
.
```

SYS04

```
DFHSIT
APPLID=SYS04
.
```

Link between SYS01 and SYS02

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

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

Link between SYS03 and SYS04

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

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

Link between SYS02 and SYS03

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

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

Indirect link from SYS04 to SYS01 routed via SYS03

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

Indirect link from SYS03 to SYS01 routed via SYS02

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

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

The terminal (SYS01)

```
DEFINE
TERMINAL(T42A)
NETNAME(XXXXX)
TYPETERM(DFHLU2)
.
```

The terminal (SYS02)

```
DEFINE
TERMINAL(T42A)
REMOTESYSTEM(PREV)
TYPETERM(DFHLU2)
.
```

The terminal (SYS03)

```
DEFINE
TERMINAL(T42A)
REMOTESYSTEM(REMT)
TYPETERM(DFHLU2)
.
```

The terminal (SYS04)

```
DEFINE
TERMINAL(T42A)
REMOTESYSTEM(REMT)
TYPETERM(DFHLU2)
.
```

The transaction (SYS01)

```
DEFINE
TRANSACTION(TRTN)
REMOTESYSTEM(NEXT)
.
```

The transaction (SYS02)

```
DEFINE
TRANSACTION(TRTN)
REMOTESYSTEM(NEXT)
.
```

The transaction (SYS03)

```
DEFINE
TRANSACTION(TRTN)
REMOTESYSTEM(NEXT)
.
```

The transaction (SYS04)

```
DEFINE
TRANSACTION(TRTN)
PROGRAM(TRNP)
.
```

Figure 53. Defining indirect links for transaction routing

Defining the direct links

The direct links between SYS01 and SYS02, SYS02 and SYS03, and SYS03 and SYS04 are MRO or APPC links defined as described in "Defining links for multiregion operation" on page 170 and "Defining APPC connections" on page 174.

Defining the indirect links

Indirect links to the terminal-owning region (TOR) can be defined to some systems in a transaction-routing path and not to others, depending on the structure of your network and how you have coded your remote terminal definitions.

For example, if an intermediate system uses hard-coded terminal definitions that do not specify REMOTESYSNET and the system does not have a direct link to the TOR, an indirect link is required. Indirect links are never required in the system to which the TOR has a direct link.

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

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

Defining the terminal

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

For the recommended methods to define remote terminals and connections to a CICS Transaction Server for z/OS system, see "Defining remote resources" on page 200.

Figure 53 on page 190 shows hard-coded remote terminal definitions that do not specify the REMOTESYSNET option. If you use these terminal definitions, the following conditions apply:

- The REMOTESYSTEM (or SYSIDNT) option in the remote terminal or connection definition must always name a link to the TOR (that is, a CONNECTION definition on which NETNAME specifies the applid of the terminal-owning region).
- The named link must be the direct link to the terminal-owning region, if one exists. Otherwise, it must be an indirect link.

Defining the transaction

For information about the definition of remote transactions, see "Defining remote resources" on page 200.

TCP/IP management and control

You can use TCP/IP management and control to monitor work that enters or leaves CICS over Transmission Control Protocol/Internet Protocol (TCP/IP) connections.

TCP/IP management and control provides, for TCP/IP networks, a subset of the management functions already provided for APPC networks and some additional functions that are not available for APPC or MRO networks.

TCP/IP networks are systems that are interconnected by the following means:

- An IPIC connection (IPCONN).
- TCP/IP connections from clients that carry, for example, Web Interface, or SOAP over HTTP requests inbound to CICS.

You can use TCP/IP management and control as follows:

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

For example, you can use CICSPlex SM, or an equivalent tool, as follows:

- You can obtain a CICSplex-wide view of the TCP/IP network.
- You can examine the following items in real time:
  - The TCP/IP network resources that a specific CICS region is using
  - The work passing in and out of a specific CICS region over the TCP/IP network
  - The CICS resources and tasks associated with a distributed transaction that flows across the CICSplex over the TCP/IP network
  - The CICS region in which a distributed transaction originated

You can save the data collected by CICS so that it can be examined offline, at some point after the tasks and resources to which it relates are no longer available.

Some useful SPI commands

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

EXEC CICS EXTRACT STATISTICS

Specify a RESTYPE of IPCONN to retrieve resource statistics for IPIC. Global statistics are not available.

EXEC CICS INQUIRE ASSOCIATION

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

EXEC CICS INQUIRE ASSOCIATION LIST

This command returns a list of tasks, in the local region, that have matching user correlation data in their associated data control blocks (ADCBs). Typically, the user correlation data has been added, at the point of origin of a distributed transaction, by a CICS XAPADMGR global user exit program. See “The XAPADMGR global user exit” on page 193.

EXEC CICS INQUIRE TASK

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

EXEC CICS PERFORM STATISTICS

Specify a statistics type of IPCONN to record resource statistics for IPIC connections. Global statistics are not available.

Socket application data (ApplData)

CICS generates 40 bytes of socket application data (ApplData) for each of the TCP sockets that it owns. CICS uses the SIOCSAPPLDATA IOCTL socket function to associate this information with the z/OS Communications Server TCP/IP socket. You can use this information to correlate TCP/IP connections with the CICS regions and transactions using them.

In CICS, you can obtain the ApplData information using the CECI INQUIRE ASSOCIATION transaction, CICSPlex SM displays, and SMF records. In TCP/IP, the ApplData information is available on the Netstat

ALL/-A, ALLConn/-a, and COnn/-c reports, and can be searched with the APPLD/-G filter. See z/OS Communications Server: IP System Administrator's Commands for additional information about using ApplData with Netstat. The ApplData information is available in the SMF 119 TCP Connection Termination record. See z/OS Communications Server: IP Configuration Reference for additional information.
The ApplData information is also available through the Network Management Interface. See z/OS Communications Server: IP Programmer's Guide and Reference for more information.

The XAPADMGR global user exit

The exit program is called, if enabled, at the attach of nonsystem tasks for which no input Origin Descriptor Record is provided.

For further information about the XAPADMGR exit, see Application association data exit in the AP domain (XAPADMGR).

CICS provides a sample global user exit program, DFH$APAD, for use at the XAPADMGR exit point. For more information about DFH$APAD, see Application associated data sample exit program: DFH$APAD.

Using CICSPlex SM to analyze TCP/IP traffic

As noted in "The XAPADMGR global user exit" on page 193, user correlation information added to the associated data origin descriptor of a task, at the point of origin of the distributed transaction, can be used as search keys for later processing carried out through CICSPlex SM.

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

?
: Matches exactly one arbitrary character

*****
: Matches zero or more arbitrary characters

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

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

For more information, see Task association information - TASKASSC.

Using CICS monitoring to analyze TCP/IP traffic

Fields 360 - 372 in the performance class monitoring records in group DFHCICS relate to TCP/IP. See Performance data in group DFHCICS.

Managing APPC connections

You can use the main terminal transaction, CEMT, to manage APPC connections. It shows how the action of the CEMT commands is affected by the way the connections have been defined to CICS.

The commands are described under the headings:

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

The commands used to achieve these actions are:

- CEMT SET CONNECTION ACQUIRED|RELEASED

- CEMT SET MODENAME AVAILABLE|ACQUIRED|CLOSED

Tip: In CICS Explorer, the ISC/MRO Connections view provides a functional equivalent to the SET CONNECTION command. See ISC/MRO Connections view in the CICS Explorer product documentation.

Detailed formats and options of CEMT commands are given in CEMT SET CONNECTION.

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

General information about managing APPC links

The operator commands controlling APPC connections cause CICS to execute many internal processes, some of which involve communication with the partner systems.

The major features of these processes are described on the following pages but you should note that the processes are sometimes independent of one another and can be asynchronous. This makes simple descriptions of them imprecise in some respects. The execution can occasionally be further modified by independent events occurring in the network, or simultaneous operator activity at both ends of an APPC connection; these circumstances are more likely when a component of the network has failed and recovery is in progress. The following sections explain the normal operation of the commands.

Note: The principles of operation described in these sections also apply to the EXEC CICS INQUIRE CONNECTION, INQUIRE MODENAME, SET CONNECTION, and SET MODENAME commands.

Acquiring a connection

The SET CONNECTION ACQUIRED command causes CICS to establish a connection with a partner system.

The major processes involved in this operation are:

- Establishing of the two LU services manager sessions in the modegroup SNASVCMG.
- Initiating of the change-number-of-sessions (CNOS) process by the partner initiating the connection.

  CNOS negotiation is executed (using one of the LU services manager sessions) to determine the numbers of contention-winner and contention-loser sessions defined in the connection. The results of the negotiation are reported in messages DFHZC4900 and DFHZC4901.
- Establishing of the sessions that carry CICS application data.

The following processes, also part of connection establishment, are described in Troubleshooting intersystem problems:

- Exchanging lognames
- Resolving and reporting synchronization information.

Connection status during the acquire process

The status of the connection before and during the acquire process is reported by the INQUIRE CONNECTION command.

Released
Initial state before the SET CONNECTION ACQUIRED command. All the sessions in the connection are released.

Obtaining
Contact has been made with the partner system, and CNOS negotiation is in progress.

Acquired
CNOS negotiation has completed for all modegroups. In this status CICS has bound the LU services manager sessions in the modegroup SNASVCMG. Some of the sessions in the user modegroups may also have been bound, either as a result of the AUTOCONNECT option on the SESSIONS definition, or to satisfy allocate requests from applications.

The results of requests for the use of a connection by application programs depend on the status of the sessions. You can control the status of the sessions with the AUTOCONNECT option of the SESSIONS definition as described in the following section.

Effects of the AUTOCONNECT option

The AUTOCONNECT attribute of the SESSIONS resource controls the acquisition of sessions in modegroups associated with the connection.

The meanings of the AUTOCONNECT attribute for APPC connections are described in "The AUTOCONNECT attribute" on page 179. Each modegroup has its own AUTOCONNECT option and the setting of this attribute affects the sessions in the modegroup:

Table 16. Effect of AUTOCONNECT on the SESSIONS resource

Setting	Effect
YES	CNOS negotiation with the partner system is performed for the modegroup, and all negotiated contention-winner sessions are acquired when the connection is acquired.
NO	CNOS negotiation with the partner system is performed, but no sessions are acquired. Contention-winner sessions can be bound individually according to the demands of application programs (for example, when a program issues an ALLOCATE command), or the SET MODENAME ACQUIRED command can be used to bind contention-winner sessions.
ALL	CNOS negotiation with the partner system is performed for the modegroup, and all negotiated sessions, contention winners, and contention losers are acquired when the connection is acquired. This setting should be necessary only on connections to non-CICS systems.

When the connection is in ACQUIRED status, the INQUIRE MODENAME command can be used to determine whether the user sessions have been made available and activated as required. The binding of user sessions is not completed instantaneously, and you may have to repeat the command to see the final results of the process.

CICS can bind contention-winner sessions to satisfy an application request, but not contention losers. However, it can assign contention-loser sessions to application requests if they are already bound. Considerations for binding contention losers are described in the next section.

Binding contention-loser sessions

Contention-loser sessions on one system are contention-winner sessions on the partner system, and should be bound by the partner.. If you want all sessions to be bound, you must make sure each side binds its contention winners.

If the connection is between two CICS systems, specify AUTOCONNECT(YES) on the SESSIONS definition for each system, or issue CEMT SET MODENAME ACQUIRED from both systems. If you are linked to a non-CICS system that is unable to send bind requests, specify AUTOCONNECT(ALL) on your SESSIONS definition.

If the remote system can send bind requests, find out how you can make it bind its contention winners so that it does so immediately after the SNASVCMG sessions have been bound.

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

Effects of the MAXIMUM option

The MAXIMUM attribute of the SESSIONS resource specifies the maximum number of sessions that can be supported for the modegroup, and the number of these that are supported as contention winners.

Operation of APPC connections is made easier if the maximum number of sessions at each end of the connection match, and the number of contention-winner sessions specified at the two ends add up to this maximum number. If this is done, CNOS negotiation does not change the numbers specified.

If the specifications at each end of the connection do not match, as has just been described, the actual values are negotiated by the LU services managers. The effect of the negotiation on the maximum number of sessions is to adopt the lower of the two values. An architected algorithm is used to determine the number of contention winners for each partner, and the results of the negotiation are reported in messages DFHZC4900 and DFHZC4901.

These results can also be deduced, as shown in Table 17 on page 196, by issuing a CEMT INQUIRE MODENAME command.

Table 17. Data displayed by INQ MODENAME

Display	**Interpretation**
MAXimum	The value specified in the sessions definition for this modegroup. This represents the true number of usable sessions only if it is equal to or less than the corresponding value displayed on the partner system.
AVAilable	Represents the result of the most recent CNOS negotiation for the number of sessions to be made available and potentially active. Following the initial CNOS negotiation, it reports the result of the negotiation of the first value of the MAXIMUM option.
ACTive	The number of sessions currently bound.

To change the MAXIMUM values, release the connection, set it OUTSERVICE, redefine it with new values, and reinstall it.

Controlling sessions with the SET MODENAME commands

The SET MODENAME commands can be used to control the sessions within the modegroups associated with an APPC connection, without releasing or reacquiring the connection.

The processes executed to accomplish this are:

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

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

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

Modegroups can normally be controlled with the few simple commands described in Table 18 on page 196.

Table 18. SET MODENAME commands

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

Table 18. SET MODENAME commands (continued)

Command	Effect
SET MODENAME CLOSED	Negotiates with the partner to reduce the available number of sessions to zero, releases the sessions, and prevents any attempt by the partner to negotiate or activate any sessions in the modegroup. Only the system issuing the command can subsequently increase the session count. Queued session requests are honored before sessions are unbound.
SET MODENAME AVAIL(maximum) ACQUIRED	If this command is issued when the modegroup is closed, the sessions are negotiated as if the connection had been newly acquired, and the contention-winner sessions are acquired. It can also be used to rebind sessions that have been lost due to errors that have caused CICS to place sessions out of service.

Command scope and restrictions

The attributes of user modegroups, which are built from a SESSIONS resource, can be changed while the modegrroup is active; the attributes of the SNASVCMG modegroup, which is built from a CONNECTION definition, cannot be modified.

The SNASVCMG modegroup is controlled by the SET CONNECTION command, or by overtyping the INQUIRE CONNECTION display data, which also affects associated user modegroups.

CEMT INQUIRE NETNAME, where the netname is the applid of the partner system, displays the status of all sessions associated with that connection, and can be useful in error diagnosis. Any attempt to alter the status of these sessions by overtyping, is suppressed.

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

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

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

Releasing the connection

The SET CONNECTION RELEASED command causes CICS to quiesce a connection and release all sessions associated with it.

The major processes involved in this operation are:

- Executing the CNOS process to inform the partner system that the connection is closing down. The number of available sessions on all modegroups is reduced to zero.
- Quiescing transaction activity using the connection. This process allows the completion of transactions that are using sessions and queued ALLOCATE requests; new requests for session allocation are refused with the SYSIDERR condition.
- Unbinding of the user and LU services manager sessions.

Connection status during the release process

Before and during the release process, the connection can be in **Acquired**, **Freeing** or **Released** state.

Acquired
Sessions are acquired; the sessions can be allocated to transactions.

Freeing
Release of the connection has been requested and is in progress.

Released
All sessions are released.

If you have control over both ends of the connection, or if your partner is unlikely to issue commands that conflict with yours, you can release the connection to quiesce activity on the connection. When the connection is in **Released** state, you can set the connection out-of-service to prevent any attempt by the partner to reacquire the connection.

The effects of limited resources

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

The connection status can be as follows:

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

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

The connection behaves in other ways exactly as for a connection over non-limited-resource links. Commands that set the modename, and release the connection operate normally.

Making the connection unavailable

The SET CONNECTION RELEASED command quiesces transactions using the connection and releases the connection.

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

If the number of available sessions for every modegroup of a connection is reduced to zero (by, for example, a CEMT SET MODENAME AVAILABLE(0) command), ALLOCATE requests are rejected. Transaction routing and function shipping requests are also rejected. The connection is effectively unavailable. However, because the remote system can renegotiate the availability of sessions and cause those sessions to be bound, you cannot be sure that this state will be held.

To prevent your partner from acquiring sessions that you have made unavailable, use the CEMT SET MODENAME CLOSED command. This reduces the number of available user sessions in the modegroup to zero and also ***locks*** the modegroup. Even if your partner now issues SET CONNECTION RELEASED followed by SET CONNECTION ACQUIRED, no sessions in the locked modegroup become bound until you specify an AVAILABLE value greater than zero.

If you lock all the modegroups, you make the connection unavailable, because the remote system can neither bind sessions nor do anything to change the state.

Having closed all the modegroups for a connection, you can go a step further by issuing CEMT SET CONNECTION RELEASED. This unbinds the SNASVCMG (LU services manager) sessions. An inquiry on the CONNECTION returns INSERVICE RELEASED (or INSERVICE FREEING if the release process is not complete).

If you now enter SET CONNECTION ACQUIRED, you free all locked modegroups and the connection is fully established. If, instead, your partner issues the same command, only the SNASVCMG sessions are bound.

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

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

```
CEMT SET MODENAME(*) CONNECTION(....) CLOSED
```

[The CONNECTION option is significant only if the MODENAME applies to more than one connection.]

```
INQ MODENAME(*) CONNECTION(....)
```

[Repeat this command until the AVAILABLE count for all non-SNAVCMG modegroups becomes zero.]

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

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

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

Figure 54. Making the connection unavailable

Allocating from APPC mode groups with no available sessions
An application program can issue ALLOCATE commands for APPC sessions that can be satisfied in either of two ways.

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

An operator can set the number of sessions in a modegroup, or close the modegroup to reduce the number of available sessions on an individual mode group to zero.

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

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

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

Diagnosing and correcting error conditions
User sessions that have become unavailable because of earlier failures can be brought back into use by restoring or increasing the *available count* with the SET MODENAME AVAILABLE(n) command. The

addition of the ACQUIRED option to this command will result in the binding of any unbound contention-winner sessions.

If the SNASVCMG sessions become unbound while user sessions are active, the connection is still acquired. A SET CONNECTION ACQUIRED command binds all contention-winner sessions in all modegroups, and may be sufficient to reestablish the SNASVCMG sessions.

Sometimes, you may not be able to recover sessions, although the original cause of failure has been removed. Under these circumstances, you should first release, then reacquire, the connection.

Summary of APPC link management

The effect of CEMT commands on the status of an APPC link are summarized.

Table 19. Effect of CEMT commands on an operational APPC link

Commands issued in the following sequence								
1	1	1						SET MODENAME AVAILABLE(0)
			1	1	1			SET MODENAME CLOSED
	2	2		2	2	1	1	SET CONNECTION RELEASED
		3			3		2	SET CONNECTION OUTSERVICE
Resulting states and reactions								
N	N	N	N	N	N	N	N	ALLOCATE requests suspended
Y	Y	N	N	N	N	Y	N	Partner can renegotiate
Y	Y	Y	Y	Y	Y	Y	Y	ALLOCATE rejected with SYSIDERR
N	Y	Y	N	Y	Y	Y	Y	SNASVCMG sessions released
—	Y	N	—	Y	N	Y	N	Partner can rebind SNASVCMG

Command scope and restrictions

The attributes of user modegroups, which are built from a SESSIONS resource, can be changed while the modegrroup is active.

The SNASVCMG modeset, on the other hand, is built from the CONNECTION definition and any attempts to modify its status with a SET or INQUIRE MODENAME command is suppressed. It is, however, controlled by the SET|INQ CONNECTION, which also affects the user modesets.

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

Defining remote resources

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

Which remote resources need to be defined?

Remote resources are resources that reside on a remote system but that need to be accessed by the local CICS system. In general, you must define all these resources in your local CICS system, in a similar way as you define your local resources, by using CICS resource definition online (RDO) or resource definition macros, depending on the resource type.

You might need to define remote resources for CICS function shipping, DPL, asynchronous processing (START command shipping), and transaction routing. No remote resource definition is required for distributed transaction processing. However, see "A note on daisy-chaining" on page 201.

The remote resources that can be defined are as follows:

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

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

A note on daisy-chaining

The descriptions of how to define remote resources in this section usually assume that there is a direct link between the local CICS and that on which the remote resource resides.

In fact, in all types of CICS intercommunication, the local and remote systems need not be directly connected. A request for a remote resource can be daisy-chained across CICS systems by defining the resource as remote in each intermediate system, as well as (where necessary) in the local system.

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

- Dynamically-routed DPL requests
- Dynamically-routed transactions started by non-terminal-related START commands
- Dynamically-routed transactions that are associated with CICS business transaction services activities.

Local and remote names for resources

CICS resources are usually referred to by name, for example, a file name for a file and a data identifier for a temporary storage queue. When you define remote resources, you must consider both the name of the resource on the remote system and the name by which it is known in the local system.

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

The following table shows local and remote resource naming. Related resources and attributes are shown by identical numbers.

Table 20. Local and remote resource naming

	CICSA (local system)	**CICSB (remote system)**
System initialization parameters	`APPLID=CICSA` 1	3 `APPLID=CICSB`
CONNECTION resources	`CONNECTION(CICR)` 2 `NETNAME(CICSB)` 3	`CONNECTION(CICL)` 1 `NETNAME(CICSA)`

FILE resources	`FILE(FILEA)` 4
	`REMOTESYSTEM(CICR)` 2
	`FILE(FILEB)`
	`FILE(local-name)`
	`REMOTESYSTEM(CICR)` 2
	`REMOTENAME(FILEB)` 5
		4 `FILE(FILEA)`
		5 `FILE(FILEB)`

The table shows two files, FILEA and FILEB, which are owned by a remote CICS system (CICSB), together with their definitions as remote resources in the local CICS system CICSA.

- FILEA has the same name on both systems, so that a reference to FILEA on either system means the same file.
- FILEB is provided with a local name on the local system, so that the file is referred to by its local name in the local system and by FILEB on the remote system. The "real" name of the remote file is specified in the REMOTENAME option. Note that CICSA can also own a local file called FILEB.

Defining remote resources for function shipping

If you use CICS function shipping, you might need to define the following remote resources: files, DL/I PSBs, transient data destinations, and temporary storage queues.

Defining remote files

A remote file is a file that resides on another CICS system. CICS file control requests that are made against a remote file are shipped to the remote system by means of CICS function shipping.

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

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

The following attributes provide CICS with sufficient information to enable it to ship file control requests to a specified remote system.

FILE(name)

The name by which the file is known on the local CICS system is specified in the FILE option. This is the name that is used in file control requests by application programs in the local system.

REMOTESYSTEM(name)

The name of the remote system to which file control requests for this file are to be shipped is specified in the REMOTESYSTEM option. If the name specified is that of the local system, the request is not shipped.

REMOTENAME(name)

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

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

RECORDSIZE(record-size)

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

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

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

KEYLENGTH(key-length)

Although MRO is supported for both user-maintained and CICS-maintained remote data tables, CICS does not allow you to define a local data table based on a remote source data set. However, there are ways around this restriction. (See File control.)

Sharing file definitions

In some circumstances, two or more CICS systems can share a common CICS system definition file (CSD). If the local and remote systems share a CSD, you need define each VSAM file used in function shipping only once.

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

For information about sharing a CSD, see Sharing user access from several CICS regions.

Defining remote DL/I PSBs

To enable the local CICS system to access remote DL/I databases, you must define the remote PSBs in a PDIR.

The form of macro used for this purpose is:

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

Figure 55. Macro for defining remote DL/I PSBs

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

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

Defining remote transient data destinations

A remote transient data destination is one that resides on another CICS system.

CICS transient data requests that are made against a remote destination are shipped to the remote system by CICS function shipping. CICS application programs can name a remote system explicitly on transient data requests, by using the SYSID option. If this is done, there is no need for the remote transient data destination to be defined on the local CICS system.

In most cases, however, applications are designed to access transient data destinations without being aware of their location, and in this case the transient data queue must be defined as a remote destination.

A remote definition provides CICS with sufficient information to enable it to ship transient data requests to the specified remote system. Specify the following attributes:

TDQUEUE(name)

REMOTESYSTEM(name)

REMOTENAME(name)

REMOTELENGTH(length)

Defining remote temporary storage queues

A remote temporary storage queue is one that resides on another CICS system. CICS temporary storage requests that are made against a remote queue are shipped to the remote system by CICS function shipping.

Applications are typically designed to access temporary storage queues without being aware of their location. In the local CICS system, you can create TSMODEL resource definitions for temporary storage queues that match a specified prefix. To make the temporary storage model point to a remote system, use the following attributes:

- REMOTEPREFIX (or XREMOTEPFX) specifies the prefix for the temporary storage queue on the remote system.

- REMOTESYSTEM specifies the name of the connection that links the local system to the remote system where the temporary storage queue resides.

When an application specifies a temporary storage queue name that matches the prefix defined by the temporary storage model, CICS ships the request to the remote system.

It is also possible for CICS application programs to name a remote system explicitly on temporary storage requests, using the SYSID option, or to use the XTSEREQ global user exit program to direct the request to a system on which the appropriate queue is defined. With these methods, there is no need for the remote temporary storage queue to be defined in the local CICS system. However, note that TSMODEL resource definitions do not support these methods for specifying a temporary storage queue that resides in a temporary storage data sharing pool. If you want to specify an explicit SYSID for a shared queue pool, in your application program or through the XTSEREQ global user exit program, you must use a temporary storage table (TST) with a TYPE=SHARED entry for the shared queue pool.

Defining remote resources for DPL

If you use CICS DPL, you might need to define remote server programs.

A remote server program is a program that resides on another CICS system. CICS program-control LINK requests that are made against a remote program are shipped to the remote system by means of CICS DPL.

Defining remote server programs

A remote server program is defined with *remote attributes* on the program definition. How you specify the attributes depends on whether DPL requests for the program are to be routed to the remote region *statically* or *dynamically*.

The *remote attributes* include:

PROGRAM(name)

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

REMOTESYSTEM(name)

REMOTENAME(name)

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

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

TRANSID(name)

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

For dynamic requests that are routed using the CICSPlex System Manager (CICSPlex SM), the TRANSID option takes on a special significance, because CICSPlex SM's routing logic is transaction-based. CICSPlex SM routes each DPL request according to the rules specified for its associated transaction. The CICSPlex SM system programmer can use the EYU9WRAM user-replaceable module to change the transaction ID associated with a DPL request.

Note: When defining your own transaction use the definition for transaction CSMI as the base for the new transaction. The transaction profile for your new transaction must specify INBFHM(ALL).

DYNAMIC(NO|YES)

Specifying remote attributes for statical routing

To route DPL requests for the program statically:

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

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

Specifying remote attributes for dynamic routing

To route DPL requests for the program dynamically:

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

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

When definitions of remote server programs aren't required

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

- The server program is to be autoinstalled.

  As an alternative to being statically defined in the client system, the remote server program can be autoinstalled when a DPL request for it is first issued. If you use this method, you need to write an autoinstall user program to supply the name of the remote system. (For details of the CICS autoinstall facility for programs, see Autoinstalling programs, map sets, and partition sets in Configuring. For programming information about writing program-autoinstall user programs, see Writing a program to control autoinstall of APPC connections.)

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

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

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

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

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

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

  Note: This assumes that the autoinstall control program *chooses* not to install a definition. If no definition is installed because autoinstall fails, the dynamic routing program is not invoked.

- The client program names the target region explicitly, by specifying the SYSID option on the EXEC CICS LINK command.

  Note:

  1. If there is no installed definition of the program named on the LINK command, the dynamic routing program is invoked but cannot route the request, which is shipped to the remote region named on the SYSID option.
  2. If the SYSID option names the local CICS region, the dynamic routing program *is* able to route the request.

- DPL calls for the server program are to be routed dynamically.

If there is no installed definition of the program named on the LINK command, the dynamic routing program is invoked and (provided that the SYSID option is not specified) can route the request.

Note: Although in some cases a remote definition of the server program may not be necessary, in others a definition will be required—to set the program's REMOTENAME or TRANSID attribute, for example. In these cases, you should install a definition that specifies DYNAMIC(YES).

Defining remote resources for asynchronous processing

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

Note, however, that an application can use the CICS RETRIEVE command to obtain the name of a remote temporary storage queue which it subsequently names in a function shipping request.

Defining remote transactions for CICS asynchronous processing

A remote transaction for CICS asynchronous processing is a transaction that is owned by another system and is invoked from the local CICS system only by START commands.

CICS application programs can name a remote system explicitly on START commands, by means of the SYSID option. If this is done, there is no need for the remote transaction to be defined on the local CICS system.

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

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

Remote transactions that are invoked only by START commands without the SYSID option require only basic information in the installed transaction definition. Specify the following attributes:

TRANSACTION(name)

REMOTESYSTEM(sysidnt)

REMOTENAME(name)

Specifies the name of the transaction as it is known in a remote system, if it is to run in a remote system or region using intersystem communication. The remote system can be another CICS region or an IMS system.

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

LOCALQ(NO|YES)

Local queuing (LOCALQ) can be specified for remote transactions that are initiated by START requests. For further details, see Asynchronous processing.

Defining remote resources for transaction routing

CICS transactions can be routed to remote regions either statically or dynamically.

A transaction that is to be routed may be started in a variety of ways:

- From a user terminal.
- By a terminal-related ATI request; for example, a terminal-related **EXEC CICS START** command.
- By a non-terminal-related ATI request; for example, by a non-terminal-related **EXEC CICS START** command.
- If the transaction is associated with a CICS business transaction services (BTS) activity, by a **BTS RUN ASYNCHRONOUS** command. For more information about BTS, see Overview of BTS.

To route these requests, you must define a routing program. CICS provides two routing programs that can route different types of request: the dynamic routing program and the distributed routing program. For more information about these programs, see Two routing programs. To route requests between CICS regions, you must specify the appropriate program in the associated system initialization parameter:

- If you use the distributed routing program, specify the **DSRTPGM** system initialization parameter in each routing and target CICS region.
- If you use the dynamic routing program, specify the **DTRPGM** system initialization parameter in each routing region.

In addition to configuring the CICS region, you must define the appropriate CICS resources:

- If the request to start the transaction is associated with a terminal, define the terminal. For more information, see "Defining terminals for transaction routing" on page 207.
- For every request, define a TRANSACTION resource with the appropriate attributes. For more information, see "Defining transactions for transaction routing" on page 214.

Defining terminals for transaction routing

Terminal-related transaction routing is the routing of transactions started from user-terminals, and transactions started by terminal-related ATI requests. There are a number of rules that define whether a terminal is eligible for transaction routing.

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

- LUTYPE6.1 connections and sessions
- MRO connections and sessions
- IBM 7770 or 2260 terminals
- Pooled 3600 or 3650 pipeline logical units
- MVS system consoles.

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

1. In the terminal-owning region:
   a. The terminal must be defined as a local resource (or must be autoinstallable).
   b. The transaction must be defined as a remote resource if it is to be initiated from a terminal or by ATI.
2. In the application-owning region:
   a. The terminal must be defined as a remote resource, unless a shipped terminal definition is available; see "Shipping terminal and connection definitions" on page 209) for more information.
   b. The transaction must be defined as a local resource.

If transaction routing requests are to be "daisy-chained" across intermediate systems, the same rules apply. In addition, both the terminal and the transaction must be defined as remote resources in the intermediate CICS systems. If you are using non-z/OS Communications Server terminals, you also need to define indirect links to the TOR on the AOR and the intermediate systems (see "Defining indirect links for transaction routing" on page 186).

Defining remote z/OS Communications Server terminals

Remote z/OS Communications Server terminals are defined with attributes that identify the path to the terminal-owing region.

Instead of defining the terminal on the application-owning region, you can arrange for a suitable definition to be shipped from the terminal-owning region when it is required. See "Shipping terminal and connection definitions" on page 209 for more information on shipping definitions.

Remote z/OS Communications Server terminals are defined using a TERMINAL resource.

- The REMOTESYSNET attribute specifies the netname (applid) of the TOR. This enables CICS to form the fully-qualified identifier of the remote terminal, even where there is no direct link to the TOR. (See "Local and remote names for terminals" on page 213.)
- The REMOTESYSTEM attribute specifies the name of the next link in the path to the TOR. If there is more than one possible path to the TOR, use REMOTESYSTEM to specify the next link in the preferred path.

  If REMOTESYSTEM names a direct link to the TOR, normally you do not need to specify REMOTESYSNET. However, if the direct link is an APPC connection to a TOR that is a member of a z/OS Communications Server generic resource group, you might need to specify REMOTESYSNET. REMOTESYSNET is needed in this case if the NETNAME specified on the CONNECTION definition is the generic resource name of the TOR (not the applid).

Only a few of the various terminal properties need be specified for a remote terminal definition. They are:

TERMINAL(trmidnt)

TYPETERM(terminal-type)

NETNAME(netname_of_terminal)

REMOTESYSTEM(sysidnt_of_next_system)

REMOTESYSNET(netname_of_TOR)

REMOTENAME(trmidnt_on_TOR)

The TYPETERM referenced by a remote terminal definition can be a CICS-supplied version for the particular terminal type, or one that you have created. If you are defining a TYPETERM that will be used **only** for remote terminals, you can ignore the **session properties**, the **paging properties**, and the **operational properties**. You can also ignore BUILDCHAIN in the **application features**.

Defining remote APPC connections
You can define a remote single-session APPC terminal using a TERMINAL and TYPETERM resource, in the same way as you would define a remote z/OS Communications Server terminal.

For more information on defining a remote z/OS Communications Server terminal, see "Defining remote z/OS Communications Server terminals" on page 207. For remote parallel-session APPC systems and devices, you must create a CONNECTION with the following attributes. A SESSIONS definition is not required for a remote connection.

CONNECTION(sysidnt_of_device)

NETNAME(netname_of_device)

REMOTESYSTEM(sysidnt_of_next_system)

REMOTESYSNET(netname_of_TOR)

REMOTENAME(sysidnt_of_device_on_TOR)

ACCESSMETHOD(VTAM)

Note: VTAM is now z/OS Communications Server.

PROTOCOL(APPC)

How to share terminal and connection definitions
In some circumstances, two or more CICS systems can share a common CICS system definition file (CSD). If the local and remote systems share a CSD, define each terminal and APPC connection only once.

Define the terminal using the TERMINAL resource, and include an associated TYPETERM resource, similar to a local terminal definition. You must specify other attributes to ensure that when the terminal is installed on the terminal-owning region, a full, local terminal definition is built. On any other system, a remote terminal definition is built:

- Specify the NETNAME of the terminal-owning region in the REMOTESYSNET attribute.
- Specify the SYSIDNT of the terminal-owning region in the REMOTESYSTEM attribute.

Similarly, an APPC connection, for example, must be fully defined using a CONNECTION resource, and must have one or more associated SESSIONS resources. Specify the NETNAME of the terminal-owning region in the REMOTESYSNET attribute and the SYSIDNT of the terminal-owning region in the REMOTESYSTEM attribute in the same way as for the terminal definition. When the connection is installed on the terminal-owning region, a connection definition is built. On any other system, a remote connection definition is built, and the SESSIONS definition is ignored.

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

Shipping terminal and connection definitions

If you are using z/OS Communications Server terminals on your terminal-owning region, you can arrange for a terminal definition to be shipped from the terminal-owning region to the application-owning region whenever it is required. If you use this method, you do not need to define the terminal on the application-owning region.

When a remote transaction is invoked from a shippable terminal, the request that is transmitted to the application-owning region is flagged to show that a shippable terminal definition is available. If the application-owning region already has a valid definition of the terminal (which might have been shipped previously), it ignores the flag. Otherwise, it asks for the definition to be shipped.

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

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

Terminal definitions can be shipped across intermediate systems. If you use shippable terminals and there is more than one possible path from the AOR to the TOR, you might want to specify the preferred path by defining indirect links to the TOR on the AOR and the intermediate systems (see “Defining indirect links for transaction routing” on page 186).

When a shipped definition is to be installed on an intermediate or application-owning region, the autoinstall user program is invoked in that region. If the name of the shipped definition clashes with that of a remote terminal or connection already installed on the region, CICS assigns an *alias* to the shipped definition, and passes the alias to the autoinstall user program. CICS-generated aliases for shipped terminals and connections are recognizable by their first character, which is always '{'. Their remaining three characters can have the values 'AAA' through '999'. Your autoinstall user program can accept a CICS-generated alias, override it, or reject the installation. Note that it can also specify an alias for a shipped definition when there is *no* clash with an installed remote definition.

You need to consider assigning aliases to shipped definitions if, for example, you have two or more terminal-owning regions that use similar sets of terminal identifiers for transaction routing to the same AOR. For information about writing an autoinstall user program to control the installation of shipped terminals, see Writing a program to control autoinstall of shipped terminals.

Shipping terminals for ATI requests

If you require a transaction that is started by ATI to acquire a remote terminal, you normally statically define the terminal to the AOR and any intermediate systems.

You do this because, for example, specifying a remote terminal for an intrapartition transient data queue (see “Defining intrapartition transient data queues” on page 222) does *not* cause a terminal definition to be shipped from the remote system. However, if a shipped terminal definition has already been received, following a previous transaction routing request, the terminal is eligible for ATI requests.

However, if the TOR and AOR are directly connected, CICS does allow you to cause terminal definitions to be shipped to the AOR to satisfy ATI requests. If you enable the user exit XALTENF in the AOR, CICS invokes this exit whenever it meets a “terminal-not-known” condition. The program you code has access to parameters, giving details of the origin and nature of the ATI request. You use these to decide the identity of the region that owns the terminal definition you want CICS to ship for you. A similar user exit, XICTENF, is available for start requests that result from EXEC CICS START.

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

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

A better way of handling terminal-related START requests is to use the enhanced routing methods described in Routing transactions invoked by START commands. If the START request is issued in the TOR, it is *not* function-shipped to the AOR: thus the “terminal-not-known” cannot occur; nor do you need to use FSSTAFF to prevent the transaction being started against the “wrong” terminal. Instead, the START executes directly in the TOR, and the transaction is routed as if it had been initiated from a terminal. If you are using shippable terminals, a terminal definition is shipped to the AOR if required.

Defining terminals as shippable
To make a terminal definition eligible for shipping, you must associate it with a TYPETERM that specifies SHIPPABLE(YES).

This method can be used for any z/OS Communications Server terminal. It is particularly appropriate if you use autoinstall in the TOR.

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

.

Defining remote non-z/OS Communications Server terminals
Non-z/OS Communications Server terminals must be defined using resource definition macros: you cannot use RDO.

A remote non-z/OS Communications Server terminal requires a full terminal control table entry in the remote system (TOR), and a terminal control table entry in the local system (AOR) that contains sufficient information about the terminal to enable CICS to perform the transaction routing. Data set control information and line information is not required for the definition of a remote terminal.

Non-z/OS Communications Server terminal definitions are not shippable.

Using resource definition macros, you can define remote non-z/OS Communications Server terminals in either of two ways:

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

Both methods allow the same terminal definitions to be used to generate the required entries in both the local and the remote system.

Definition using DFHTCT TYPE=REMOTE

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

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

Figure 56. Defining a remote non-z/OS Communications Server terminal (transaction routing)

SYSIDNT specifies the name of the connection to the terminal-owning region. If there is no direct link to the TOR, SYSIDNT must specify the name of an **indirect link** (see "Defining indirect links for transaction routing" on page 186).

Sharing terminal definitions

This section applies to all supported types of non-z/OS Communications Server terminals.

With the exception of SYSIDNT, the operands of DFHTCT TYPE=REMOTE form a subset of those that can be specified with DFHTCT TYPE=TERMINAL. Any of the remaining operands can be specified. They are ignored unless the SYSIDNT operand names the local system, in which case the macro becomes equivalent to the DFHTCT TYPE=TERMINAL form.

A single DFHTCT TYPE=REMOTE macro can therefore be used to define the same terminal in both the local and the remote systems. A typical use of this method of definition is shown in Figure 57 on page 211.

```
  Local System CICL                  Remote System CICR
        AOR                                 TOR

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

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

DFHTCT TYPE=REMOTE,                DFHTCT TYPE=REMOTE,
       SYSIDNT=CICR                       SYSIDNT=CICR
       TRMIDNT=aaaa,                      TRMIDNT=aaaa,
       TRMTYPE=3277,                      TRMTYPE=3277,
       TRMMODL=2,                         TRMMODL=2,
       ALTSCRN=(43,80)                    ALTSCRN=(43,80)
       .                                   .
       .                                   .
DFHTCT TYPE=FINAL                  DFHTCT TYPE=FINAL
```

Figure 57. Typical use of DFHTCT TYPE=REMOTE macro

Note: VTAM is now z/OS Communications Server.

In Figure 57 on page 211, the same terminal definition is used in both the local and the remote systems.

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

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

The terminal identification is "aaaa" in both systems.

Definition using DFHTCT TYPE=REGION

If you use the DFHTCT TYPE=REGION macro, you can define remote terminals in the same way as local terminals, using DFHTCT TYPE=SDSCI, TYPE=LINE, and TYPE=TERMINAL macros.

The definitions must, however, be preceded by a DFHTCT TYPE=REGION macro, which has the following form:

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

SYSIDNT specifies the name of the connection to the terminal-owning region. If there is no direct link to the TOR, SYSIDNT must specify the name of an **indirect link** (see "Defining indirect links for transaction routing" on page 186).

Sharing terminal definitions

If SYSIDNT does not name the local system, only the information required to build a remote terminal entry is extracted from the succeeding definitions. DFHTCT TYPE=SDSCI and TYPE=LINE definitions are ignored. Parameters of TYPE=TERMINAL definitions that are not part of the TYPE=REMOTE subset are also ignored.

A return to local system definitions is made by using DFHTCT TYPE=REGION,SYSIDNT=LOCAL.

A typical use of this method of definition is shown in Figure 58 on page 212.

```
 Terminal-Owning Region        Application-Owning Region

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

                                  DFHTCT TYPE=REGION,
                                         SYSIDNT=TERM

   COPY TERMDEFS                  COPY TERMDEFS

                                  DFHTCT TYPE=REGION,
                                         SYSIDNT=LOCAL

   DFHTCT TYPE=FINAL              DFHTCT TYPE=FINAL
```

Figure 58. Typical use of DFHTCT TYPE=REGION macro

In Figure 58 on page 212, the same copy book of terminal definitions is used in both the terminal-owning region and the application-owning region.

In the terminal-owning region, local terminal entries are built.

In the application-owning region, the fact that the sysidnt specified in the TYPE=REGION macro differs from the sysidnt specified in the DFHTCT TYPE=INITIAL macro causes remote terminal entries to be built.

Local and remote names for terminals
CICS uses a unique identifier for every terminal that is involved in transaction routing. The identifier is formed from the applid (netname) of the CICS system that owns the terminal and the terminal identifier specified in the terminal definition on the terminal-owning region.

If, for example, the applid of the CICS system is PRODSYS and the terminal identifier is L77A, the fully-qualified terminal identifier is PRODSYS.L77A.

The following rules apply to all forms of hard-coded remote terminal definitions:

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

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

Providing the netname of the TOR
You must always ensure that the remote terminal definition allows CICS to access the netname of the TOR.

In the following examples, it is assumed that the applid of the terminal-owning region is PRODSYS.

Remote terminal definition		

z/OS Communications Server terminal definition with direct link to TOR	TERMINAL resource specifies
	`REMOTESYSTEM(PD1)`
		CONNECTION resource specifies
		`CONNECTION(PD1)`
		`NETNAME(PRODSYS)`

z/OS Communications Server terminal definition with no direct link to TOR	TERMINAL resource specifies
	`REMOTESYSTEM(NEXT)`
	`REMOTESYSNET(PRODSYS)`
		CONNECTION resource specifies
		`CONNECTION(NEXT)`
		`NETNAME(INTER1)`

Non-z/OS Communications Server terminal definition with direct link to TOR (method 1)	`DFHTCT TYPE=REMOTE,`
	`SYSIDNT=PD1`
		CONNECTION resource specifies
		`CONNECTION(PD1)`
		`NETNAME(PRODSYS)`

Non-z/OS Communications Server terminal definition with direct link to TOR (method 2)	`DFHTCT TYPE=REGION,`
	`SYSIDNT=PD1`
		CONNECTION resource specifies
		`CONNECTION(PD1)`
		`NETNAME(PRODSYS)`

Non-z/OS Communications Server terminal definition with no direct link to TOR (method 1)	`DFHTCT TYPE=REMOTE,`
	`SYSIDNT=REMT,`
	`DFHTCT TYPE=TERMINAL,`
	`...`
		CONNECTION resource specifies
		`CONNECTION(REMT)`
		`NETNAME(PRODSYS)`
		`ACCESSMETHOD(INDIRECT)`
		`INDSYS(NEXT)`

Terminal aliases
The name by which a terminal is known in the application-owning region is usually the same as its name in the terminal-owning region. You can, however, choose to call the remote terminal by a different name (an alias) in the application-owning region.

You have to provide an alias if the terminal-owning region and the application-owning region each own a terminal with the same name; you cannot have a local terminal definition and a remote terminal definition with the same name. (Nor can you have two remote terminal definitions (for terminals on different remote regions) with the same name.)

If you use an alias, you must also specify the “real” name of the terminal as its remote name, as follows:

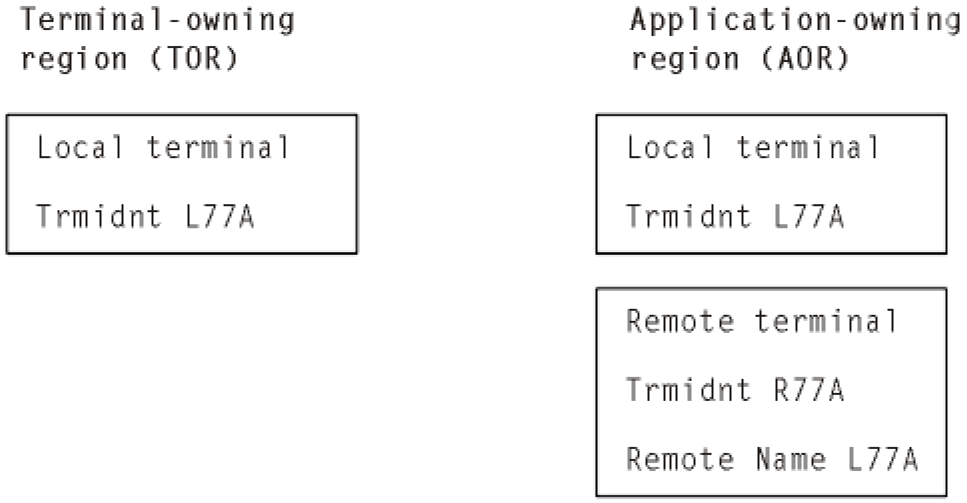


Figure 59. Local and remote names for remote terminals

You specify the remote name in the REMOTENAME attribute of the TERMINAL resource.

Defining transactions for transaction routing

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

1. When an EXEC CICS START command uses the SYSID option to name the remote region on which the transaction is to run, a remote region named explicitly on in the SYSID option takes precedence over one named on the transaction definition.
2. The remote attributes specify DYNAMIC(NO), and the REMOTESYSTEM name is either blank or the sysid of the local system.

   In this case, the transaction is executed locally, and transaction routing is not involved.
3. The remote attributes specify DYNAMIC(NO), and the REMOTESYSTEM name differs from the sysid of the local system.

   In this case, the transaction is routed to the system named in the REMOTESYSTEM option. This is known as **static** transaction routing. The REMOTESYSTEM option must name a **direct** link to another system (not an indirect link nor a remote APPC connection).
4. The remote attributes specify DYNAMIC(YES).

   In this case, the decision about where to execute the transaction is taken by your dynamic or distributed routing program. See Two routing programs.

   Note: Exceptions to this rule are transactions initiated by EXEC CICS START commands that are ineligible for enhanced routing. For example, if one of these transactions is defined as DYNAMIC(YES), your dynamic routing program is invoked but cannot route the transaction. See Routing transactions invoked by START commands.

The name in the TRANSACTION option is the name by which the transaction is invoked in the local region. TASKREQ can be specified if special inputs, such as a program attention (PA) key, program function (PF) key, light pen, magnetic slot reader, or operator ID card reader, are used.

If there is a possibility that the transaction will be executed locally, the definition must follow the normal rules for the definition of a local transaction. In particular, the PROGRAM option must name a user program that will be installed in the local system. When the transaction is routed to another system, the program associated with it is always the relay program DFHAPRT, irrespective of the name specified in the PROGRAM option.

The PROFILE option names the profile that is to be used for communication between the terminal and the relay transaction (or the user transaction if the transaction is executed locally). For remote execution, the TRPROF option names the profile that is to be used for communication on the session between the relay transaction and the remote transaction-owning system. Information about profiles is given under "Defining communication profiles" on page 218.

When a transaction will always be routed to a remote system, so that the transaction executed in the local system is always the relay transaction, you might want to specify some options for control of the relay transaction:

- You can set or default TWASIZE to zero, because the relay transaction does not require a TWA.
- You should specify transaction security for routed transactions that are operator initiated. You do not need to specify resource security checking, because the relay transaction does not access resources. See Transaction security for information on security.
- For transaction routing on mapped APPC connections or MRO sessions, you should code the RTIMOUT option on the communication profile named on the TRPROF option of the transaction definition. This causes the relay transaction to be timed out if the system to which a transaction is routed does not respond within a reasonable time.

  Deadlock timeout (specified on the DTIMOUT option of the transaction definition) is not triggered for terminal I/O waits. Because the relay transaction does not access resources after obtaining a session, it has little need for DTIMOUT except to trap suspended ALLOCATE requests. (Methods for specifying whether, if there are no free sessions to a remote system, ALLOCATE requests should be queued or rejected, are described in Intersystem session queue management.)

The method you use to define transactions for routing may differ, depending on whether the transactions are to be statically or dynamically routed.

Static transaction routing
There are two methods of defining transactions that are to be statically routed.

Using separate local and remote definitions
You create a remote definition for the transaction, and install it on the requesting region: the REMOTESYSTEM option must specify the name of the target region (or the name of an intermediate system, if the request is to be "daisy-chained").

You install separate remote definitions for the transaction on any intermediate systems: the REMOTESYSTEM option must specify the name of the next system in the routing chain. You create a local definition for the transaction, and install it on the target region: the REMOTESYSTEM option must be blank, or specify the name of the target region.

If the transaction may be initiated by an EXEC CICS START command, check whether you can use the enhanced routing method described in Routing transactions invoked by START commands. If enhanced routing is possible, define the transaction as ROUTABLE(YES) in the region in which the START will be issued.

If two or more systems along the transaction-routing path share the same CSD, the transaction definitions should be in different groups.

Using dual-purpose definitions
A dual-purpose transaction definition is shared between the requesting region and the target region (and possibly between intermediate systems too, if "daisy chaining" is involved). The REMOTESYSTEM attribute specifies the name of the target region.

If the transaction may be initiated by an EXEC CICS START command, check whether you can use the enhanced routing method described in Routing transactions invoked by START commands. If enhanced routing is possible, specify ROUTABLE(YES).

When the definition is installed on each system, the local CICS compares its SYSIDNT with the REMOTESYSTEM name. If they are different (as in the requesting region), a remote transaction definition is created. If they are the same (as in the target region), a local transaction definition is installed.

It is recommended that, for static transaction routing, you use this method wherever possible. Because you have only one set of CSD records to maintain, it provides savings in disk storage and time. However, you can use it only if your systems share a CSD. For information about sharing a CSD, see Multiple users of the CSD within a CICS region (non-RLS).

Dynamic transaction routing
There are three methods of defining transactions that are to be dynamically routed.

Note: Using dual-purpose definitions (on which the REMOTESYSTEM option specifies the default target region) is a fourth possible method, but is not recommended for transactions that are to be dynamically routed. This is because the DYNAMIC(YES) attribute on the shared definition causes the dynamic routing program to be invoked unnecessarily in the target region, after the transaction has been routed.

Using separate local and remote definitions
This is the recommended method for transactions that may be initiated by terminal-related EXEC CICS START commands.

This method is as described under “Static transaction routing” on page 215.

For dynamic routing of a transaction initiated by a START command, you must define the transaction as ROUTABLE(YES) in the region in which the START command is issued.

Using identical definitions
This is the recommended method for transactions that are associated with CICS business transaction services (BTS) activities, and transaction that can be initiated by non-terminal-related START commands.

These types of transactions are routed using the distributed routing model, which is a peer-to-peer system; each region can be both a requesting/routing region and a target region. Therefore, the transactions should be defined identically in each participating region. The regions might or might not be able to share a CSD. For more information, see Sharing user access from several CICS regions.

On each TRANSACTION definition:

- Specify DYNAMIC(YES).
- Do not specify a value for the REMOTESYSTEM option.
- If the transaction can be initiated by a non-terminal-related START command, specify ROUTABLE(YES).

The "identical definitions" method differs from the "dual-purpose definitions" method in several ways:

- It is used for dynamic, not static, routing.
- The TRANSACTION definitions do not specify the REMOTESYSTEM option.
- The participating regions are not required to share a CSD.

Using a single transaction definition in the TOR
This is the recommended method for terminal-initiated transactions.

Using it, in the TOR (and in any intermediate systems) you install only *one* transaction definition that specifies DYNAMIC(YES). This single definition provides a set of default attributes for *all* transactions that are dynamically routed. The name of the common definition is that specified on the DTRTRAN system initialization parameter. The default name is CRTX, which is the name of a CICS-supplied transaction definition that is included in the CSD group DFHISC.

If, at transaction attach, CICS cannot find an installed resource definition for a user transaction identifier (transid), it attaches a transaction built from the user transaction identifier and the set of attributes taken from the common transaction definition. (If the transaction definition specified on the DTRTRAN parameter is not installed, CICS attaches the CICS-supplied transaction CSAC. This sends message DFHAC2001—“Transaction *'tranid'* is unrecognized”—to the user's terminal.) Because the common transaction definition specifies DYNAMIC(YES), CICS invokes the dynamic transaction routing program to select a target application-owning region and, if necessary, name the remote transaction.

In the target AOR, you install a local definition for each dynamically-routed transaction.

If you use this method for all your terminal-initiated transactions:

- Dynamically-routed transactions should be installed in the terminal-owning region (if local to the TOR), or the application-owning region (if local to the AOR), but not both.
- The only terminal-initiated transaction you should define as dynamic is the dynamic transaction routing definition specified on the DTRTRAN parameter.
- The only terminal-initiated transactions you should define as remote are those that are to be statically routed.

This greatly simplifies the task of managing resource definitions.

It is recommended that you create your own common transaction definition for dynamic routing, using CRTX as a model. The definition is supplied in RDO group DFHISC, with the following attributes:

DTIMOUT(NO)

DYNAMIC(YES)
This is required for a dynamic transaction routing definition that is specified on the DTRTRAN system initialization parameter. You can change the other parameters when creating your own definition, but must specify DYNAMIC(YES).

INDOUBT(BACKOUT)

PROFILE(DFHCICST)

PROGRAM(########)
The CICS-supplied default transaction specifies a dummy program name, ########. If your dynamic transaction routing program allows a transaction to run in the local region, and its definition specifies the dummy program name, CICS is unlikely to find such a program, causing a "program-not-found" condition.

You are recommended to specify the name of a program that you want CICS to invoke whenever the transaction:

- Is not routed to a remote system, and
- Is not rejected by the dynamic transaction routing program by means of the DYRDTRRJ parameter, and
- Is run in the local region.

You can use the local program to issue a suitable response to a user's terminal if the dynamic routing program decides it cannot route the transaction to a remote system.

REMOTENAME()

SPURGE(YES)

STATUS(ENABLED)

TASKDATALOC(ANY)

TASKDATAKEY(CICS)

TPURGE(YES)

TRANSACTION(CRTX)
The name of the CICS-supplied dynamic transaction routing definition. Change this to specify your own transaction identifier.

TRPROF(DFHCICSS)

TWASIZE(00000)

RESTART(NO)
This attribute is forced for a routed transaction.

REMOTESYSTEM
You can code this to specify a default AOR for transactions that are to be dynamically routed.

ROUTABLE(NO)
This attribute relates to the enhanced routing of transactions initiated by EXEC CICS START commands.

Specifying ROUTABLE(YES) means that, if the transaction is the subject of an eligible START command, it will be routed using the enhanced routing method described in Routing transactions invoked by START commands. You are recommended to:

- Specify ROUTABLE(NO) on the common transaction definition
- Install individual definitions of transactions that may be initiated by START commands.

By reserving the common definition for use with transactions that are started from user-terminals, you prevent transactions that are initiated by terminal-related START commands from being dynamically routed “by accident”.

Defining remote resources for DTP

For MRO and LUTYPE6.1 links, there is no need to define any remote resources for DTP, provided that the front-end and back-end systems are directly connected. Both the remote system and the remote transaction are identified on the EXEC CICS commands issued by the front-end transaction. CICS therefore has all the necessary information to connect a session and attach the back-end transaction.

However, if the back-end transaction is to be routed to, it must be defined as a remote resource on the intermediate systems—see “A note on daisy-chaining” on page 201.

If you use the EXEC CICS API over APPC links, you can either identify the remote system and transaction explicitly, as for MRO and LUTYPE6.1 links, or by reference to a PARTNER resource. If you choose to do the latter, you need to create the appropriate PARTNER definitions. If you use the CPI Communications API over APPC links, the syntax of the commands *requires* you to create a PARTNER definition for every remote partner referenced.

Specify the following attributes for the PARTNER resource:

PARTNER(sym_dest_name)

NETWORK(name)
This attribute is optional

NETNAME(name)

PROFILE(name)
This attribute is optional

TPNAME(name)
XTPNAME(value)
Specify TPNAME or XTPNAME, but not both.

The PARTNER resource has been designed specifically to support **Systems Application Architecture (SAA) conventions**. For more guidance about this, see the z/VM: CPI Communications User's Guide.

Defining local resources

This section discusses how to define resources, required for intersystem communication, that reside in the local CICS system.

Defining communication profiles

When a transaction acquires a non-IPIC session to another system, either explicitly with an ALLOCATE command or implicitly because it uses, for example, non-IPIC function shipping, a communication profile is associated with the communication between the transaction and the session.

The communication profile specifies the following information:

- Whether function management headers (FMHs) received from the session are to be passed on to the transaction.
- Whether input and output messages are to be journaled, and if so the location of the journal.
- The node error program (NEP) class for errors on the session.

- For APPC sessions, the modename of the group of sessions from which the session is to be allocated. If the profile does not contain a modename, CICS selects a session from any available group.

CICS provides a set of default profiles, which it uses for various forms of communication. Also, you can define your own profiles, and name a profile explicitly on an ALLOCATE command.

A profile is always required for a session acquired by an ALLOCATE command; either a profile that you have defined and which is named explicitly on the command, or the default profile DFHCICSA. If CICS cannot find the profile, the CBIDERR condition is raised in the application program. Profiles are only required for non-IPIC communication.

For IPIC communication RTIMOUT defines the limit for how long a task will wait for a response message once it has successfully transmitted a request message. It is defined in the PROFILE associated with the TRANSACTION resource definition that a task is using, see PROFILE resources. DTIMOUT is taken from the TRANSACTION resource definition, see TRANSACTION resources. DTIMOUT controls how long a task is suspended for while it is enqueued waiting for use of the buffer that the TCP/IP stack provides to allow tasks to send message into the network on a particular socket. It also sets a time limit on how long a task will wait in the session allocation queue. Each IPCONN has a pool of sessions which get allocated to new tasks that want to send messages over the connection. Once a task is allocated a session then it is held until the task terminates, or issues a syncpoint. The session allocation queue is there to allow new tasks to wait when there are no free sessions. DTIMOUT for IPIC will trigger if a task is in a SOCKET wait or ENQUEUE state waiting for a socket, where as RTIMOUT will trigger when the task has transmitted data to the network.

The following attributes of a PROFILE resource are relevant to intersystem sessions:

PROFILE(name)

MODENAME(name)
This attribute is optional

INBFMH(NO|ALL)
This attribute is optional.

For MRO sessions that are acquired by an ALLOCATE command, CICS always uses INBFMH(ALL), no matter what is specified in the profile.

For APPC conversations, this attribute is ignored; APPC FMHs are never passed to CICS application programs.

JOURNAL(NO|value)
This attribute is optional

MSGJRNL(NO|INPUT|OUTPUT|INOUT)
This attribute is optional

NEPCLASS(0|value)
This attribute is optional

RTIMOUT(NO|value)
This attribute is optional.

It is usually important to ensure that an intercommunicating transaction never waits indefinitely for data from its partner transaction. The RTIMOUT attribute should be given a value suitable for intersystem working: rather less than the timeout periods typically specified for terminals used as operator interfaces. The RTIMOUT value should also be greater than the DTIMOUT value specified on the partner transaction definition.

Communication profiles for principal facilities

A profile is also associated with the communication between a transaction and its principal facility. You can name the profile when you define the TRANSACTION resource, or you can allow the default to be taken. The PROFILE for a principal facility profile has more options than the PROFILE for alternate facilities.

The RTIMOUT value defined for a back-end transaction needs to be at least as great as that specified for its front-end partner's principal facility. This is to cover the possibility of the back-end transaction waiting almost that period of time (plus some execution and network time) to receive data from its front-end.

Default profiles

CICS provides a set of communication profiles that it uses when the user does not, or cannot, specify a profile explicitly.

DFHCICSA INBFMH(ALL)
The default profile for alternative facilities that are acquired with an application program ALLOCATE command. A different profile can be named explicitly on the ALLOCATE command.

This profile is also used as a principal facility profile for some CICS-supplied transactions.

DFHCICSC
The profile for principal facilities of the CICS CISC and CISS transactions, which perform IPCONN acquire processing. DFHCICSC contains the parameter **`RTIMOUT(0030)`**, so that requests to transmit IPIC capability exchange messages during IPCONN acquire processing are subject to a read timeout of 30 seconds.

DFHCICSE
The error profile for principal facilities. CICS uses this profile to pass an error message to the principal facility when the required profile cannot be found.

DFHCICSF INBFMH(ALL)
The profile that CICS uses for the session to the remote system or region when a CICS application program issues a function shipping or DPL request.

DFHCICSP
The profile for principal facilities of the CICS-supplied page-retrieval transaction, CSPG. CICS uses this profile for CSPG even if you alter the CSPG transaction definition to specify a different one. For further information about communication profiles used by CICS-supplied transactions, see CSPG - page retrieval.

DFHCICSR INBFMH(ALL)
The profile that CICS uses in transaction routing for communication between the user transaction (running in the transaction-owning region) and the interregion link or APPC link.

Note that the user-transaction's principal facility is the surrogate TCTTE in the transaction-owning region, for which the default profile is DFHCICST.

DFHCICSS INBFMH(ALL)
The profile that CICS uses in transaction routing for communication between the relay transaction (running in the terminal-owning region) and the interregion link or APPC link.

DFHCICST
The default profile for principal facilities. You can specify a different profile for a particular transaction with the PROFILE attribute of the TRANSACTION resource.

DFHCICSV
The profile for principal facilities of the CICS-supplied transactions CSNE, CSLG, and CSRS. It is the same as DFHCICST, except that DVSUPRT(VTAM) is specified in place of DVSUPRT(ALL).

You should not modify this profile.

Note: VTAM is the previous name for z/OS Communications Server.

Modifying the default profiles

You can modify a default profile.

A typical reason for modification is to include a modename to provide class of service selection for, say, function shipping requests on APPC links. If you do this, you must ensure that every APPC link in your installation has a group of sessions with the specified modename.

You must not modify DFHCICSV, which is used exclusively by some CICS-supplied transactions.

You can modify DFHCICSP, used by the CSPG page-retrieval transaction. The supplied version of DFHCICSP specifies **UCTRAN(YES)**. Be aware that, if you specify **UCTRAN(NO)**, terminals defined with **UCTRAN(NO)** will be unable to make full use of page-retrieval facilities.

If you modify DFHCICSA, you must retain **INBFMH(ALL)**, because it is required by some CICS-supplied transactions. Modifying this profile does not affect the profile options assumed for MRO sessions.

You can modify DFHCICSF, used for function shipping and DPL requests over MRO or SNA. One reason for doing so might be to set a value of the **RTIMOUT** option. This option defaults to NO, which means that an intercommunicating transaction waits indefinitely for data from its partner transaction.

Architected processes

An architected process is an IBM-defined method of allowing dissimilar products to exchange intercommunication requests in a way that is understood by both products.

For example, a typical requirement of intersystem communication is that one system should be able to schedule a transaction for execution on another system. Both CICS and IMS have transaction schedulers, but their implementation differs considerably. The intercommunication architecture overcomes this problem by defining a model of a “universal” transaction scheduling process. Both products implement this architected process, by mapping it to their own internal process, and are therefore able to exchange scheduling requests.

The architected processes implemented by CICS are:

- System message model—for handling messages containing various types of information that needs to be passed between systems (typically, DFS messages from IMS)
- Scheduler model—for handling scheduling requests
- Queue model—for handling queuing requests (in CICS terms, temporary-storage or transient-data requests)
- DL/I model—for handling DL/I requests
- LU services model—for handling requests between APPC service managers.

Note: With the exception of the APPC LU services model, the architected processes are defined in the LUTYPE6.1 architecture. CICS, however, also uses them for function shipping on APPC links by using APPC migration mode.

The appropriate models are also used for CICS-to-CICS communication. The exceptions are CICS file control requests, which are handled by a CICS-defined file control model, and CICS transaction routing, which uses protocols that are private to CICS.

During resource definition, your only involvement with architected processes is to ensure that the relevant transactions and programs are included in your CICS system, and possibly to change their priorities.

Process names

Architected process names are one through four bytes long, and have a first byte value that is less than X'40'.

In CICS, the names are specified as four-byte hexadecimal transaction identifiers. If CICS receives an architected process name that is less than four bytes long, it pads the name with null characters (X'00') before searching for the transaction identifier.

CICS supplies the processes shown in Figure 60 on page 222.

```
  XTRANID     TRANSID     PROGRAM     DESCRIPTION
For CICS file control
     -        CSMI        DFHMIRS     File control model

For LUTYPE6.1 architected processes
  01000000    CSM1        DFHMIRS     System message model
  02000000    CSM2        DFHMIRS     Scheduler model
  03000000    CSM3        DFHMIRS     Queue model
  05000000    CSM5        DFHMIRS     DL/I model

For APPC architected processes
  06F10000    CLS1        DFHZLS1     LU services model
  06F20000    CLS2        DFHLUP      LU services model
     -        CLS3        DFHLUP      LU services model
```

Figure 60. CICS architected process names

Modifying the architected process definitions

You can modify any of the definitions for the architected processes. In particular, you may want to change the DTIMOUT value on the mirror transactions.

The previous list shows that the CICS file control model and the architected processes for function shipping all map to program DFHMIRS, the CICS mirror program. The inclusion of different transaction names for the various models enables you to modify some of the transaction attributes. You must not, however, change the XTRANID, TRANSID, or PROGRAM values.

The definitions for the mirror transactions are supplied with DTIMOUT(NO) specified. If you are uncomfortable with this situation, you should change the definitions to specify a value other than NO on the DTIMOUT option.

Interregion function shipping

Function shipping using MRO or IPIC connectivity can employ long-running mirror tasks. Function shipping using MRO can also use the short-path transformer program.

(See Long-running mirror tasks for MRO, Long-running mirror tasks for IPIC, and The short-path transformer for MRO.)

If you modify one or more of the mirror transaction definitions, you must evaluate the effect that this can have on interregion function shipping.

The short-path transformer always specifies transaction CSMI. It is not, however, used for DL/I requests; they arrive as requests for process X'05000000', corresponding to transaction CSM5.

Selecting required resource definitions for installation

The profiles and architected processes, and other transactions and programs that are required for ISC, IPIC, and MRO, are contained in the IBM protected groups DFHSTAND, DFHISC and DFHISCIP.

About this task

For information about how to include these pregenerated groups in your CICS system, see Supplied resource definitions, groups, and lists.

Install the following CICS supplied CSD groups for intercommunication:

- For MRO and ISC connections, you must install groups DFHSTAND and DFHISC.
- For IPIC connections, you must install groups DFHSTAND, DFHISC, and DFHISCIP.

Defining intrapartition transient data queues

The attributes that apply to queues that cause automatic transaction initiation, or that specify an associated principal facility (such as a terminal or another system), in an intercommunications environment are listed.

Specify the following attributes:

TDQUEUE(name)

TYPE(Intra)

ATIFACILITY(terminal)

RECOVSTATUS(logical)

FACILITYID (terminal)

RECOVSTATUS(name)

TRANSID

TRIGGERLEVEL(value)

USERID(userid)

WAIT(yes)

WAITACTION(reject)

Transactions

A transaction that is initiated by an intrapartition transient data queue must reside on the same system as the queue. That is, the transaction that you name in the queue definition must not be defined as a remote transaction.

Principal facilities

The principal facility that is to be associated with a transaction started by automatic transaction initiation (ATI) is specified in the transient data queue definition. A principal facility can be one of the following:

A local terminal

A local terminal is a terminal that is owned by the same system that owns the transient data queue and the transaction.

For any local terminal other than an APPC terminal, you need to specify a destination of terminal, and give a terminal identifier. If you omit the terminal identifier, the name of the terminal defaults to the name of the queue.

A remote terminal

A remote terminal is a terminal that is defined as remote on the system that owns the transient data queue and the associated transaction.

Automatic transaction initiation with a remote terminal is a form of CICS transaction routing (see CICS transaction routing), and the normal transaction routing rules apply.

For any remote terminal other than an APPC terminal, specify a destination of terminal and a terminal identifier.

The terminal itself must be defined as a remote terminal (or a shipped terminal definition must be made available), and the terminal-owning region must be connected to the local system either by an IRC link or by an APPC link.

A local session or APPC device

You can name a local connection definition in the definition for the transient data queue. The remote system can be connected by IRC, LUTYPE6.1, or APPC link. In the APPC case, "system" can be a hard-coded terminal-like device.

CICS allocates a session on the specified system, which becomes the principal facility to **transid**. The transaction program converses across the session using the appropriate DTP protocol.

The transaction starts in 'allocated' state on its principal facility. Then it identifies its partner transaction; that is, the process to be connected to the other end of the session. In the APPC protocol, it does this by issuing the **`EXEC CICS CONNECT PROCESS`** command, a command normally only used to start a conversation on an alternate facility.

The partner transaction, having been started in the back end with the conversation in receive state, also sees the session as its principal facility. This is unusual in that CICS treats either end of the session as a principal facility. On both sides, the conversation identifier is taken from EIBTRMID if needed, but it is also implied on later commands, as is the case for principal facilities.

A remote APPC session or device

A remote connection is defined as remote on the system that owns the transient data queue and the associated transaction.

Automatic transaction initiation with a remote APPC connection is a form of CICS transaction routing and the normal transaction routing rules apply.

You can name a remote connection in the definition for the transient data queue.

The connection itself must be defined as a remote connection (or a shipped connection definition must be made available), and the terminal-owning region must be connected to the local system either by an IRC link or by an APPC link. The remarks in "A local session or APPC device" on page 223 about handling the link after transaction initiation apply also to routed transactions.

Defining local resources for DPL

To support DPL, special resource definitions are sometimes necessary for server programs and mirror transactions.

Mirror transactions

You can specify whatever names you like for the mirror transactions to be initiated by DPL requests. Each of these transaction names must be defined in the server region on a transaction that invokes the mirror program DFHMIRS.

Defining user transactions to invoke the mirror program gives you the freedom to specify appropriate values for all the other options on the transaction resource definition.

The user transaction when defined as remote will only work within an APPC configuration. Within an IPIC, EXCI or MRO configuration, the mirror transaction must run in the local CICS region, so it must be defined with DYNAMIC(NO) and no REMOTE attributes. Routing the mirror transaction to another CICS region can impact performance and make problem determination more difficult.

Server programs

If a local program is to be requested by some other region as a DPL server, there must be a resource definition for that program.

The definition can be statically defined, or installed automatically (autoinstalled) when the program is first called. (For details of the CICS autoinstall facility for programs, see Autoinstalling programs, map sets, and partition sets in Configuring.)

Where is data converted?

When CICS intercommunication uses SNA links, system data is transmitted in EBCDIC format. Therefore, ASCII-based systems convert all data except for application data areas, which are converted by the system that receives the data.

Data conversion for function shipping and DPL

For function shipping and DPL, data can be converted in the ASCII-based system or in CICS Transaction Server for z/OS.

For function shipping and DPL from an ASCII-based system to CICS Transaction Server for z/OS, the ASCII-based system converts the resource names, and CICS Transaction Server for z/OS converts the user data.

Table 21. Data conversion for function shipping and DPL

Request type	Data	Conversion type	Where converted
TS	Queue name	Character	ASCII system
TS	FROM area	As specified in DFHCNV table	Receiving system
TD	Queue name	Character	ASCII system
TD	INTO area	As specified in DFHCNV table	Receiving system
FC	File name	Character	ASCII system
FC	SET area	As specified in DFHCNV table	Receiving system
FC	Key	As specified in DFHCNV table	Receiving system
IC	Transaction ID	Character	ASCII system
IC	FROM area	As specified in DFHCNV table	Receiving system
IC	RTERMID, RTRANSID, REQID	Character	ASCII system
PC	Program name	Character	ASCII system
PC	COMMAREA	As specified in DFHCNV table	Receiving system

For function shipping and DPL to an ASCII-based system from CICS Transaction Server for z/OS, the ASCII-based system converts all the data.

Conversion of application data is done field-by-field. Thus, ensure that the size of each field in the application data is sufficient to hold the result of the conversion applied to it. (This is particularly relevant where a field in the application data contains both SBCS and DBCS characters).

Data conversion for distributed transaction processing

In distributed transaction processing, all data areas are managed by the application, and therefore data conversion is the application's responsibility.

When you design your applications, you can choose to convert data in CICS Transaction Server for z/OS, in the ASCII-based system, or in both.

Data conversion for transaction routing

CICS Transaction Server for z/OS does not convert data for transaction routing.

Screen data always flows as 3270 data streams. COMMAREAs and TCTUAs (which are relevant to pseudoconversational transactions) are converted by the ASCII system.

Avoiding data conversion

In many cases, you can design your applications to reduce the amount of data that is converted.

For example, if an EBCDIC-based system acts as a file manager for an ASCII-based system, you can avoid converting any data by using ASCII to encode the data in the file.

Conversely, if data is held in the ASCII-based system purely for the purpose of communicating with an EBCDIC-based system, you can avoid converting the data by coded it in EBCDIC.

Types of conversion

The possible types of conversion are standard conversion, no conversion, and user-defined nonstandard conversion.

Standard conversion

This applies to:

- Single-byte character sets (SBCS)
- Graphic or double-byte character sets (DBCS)
- Mixed character sets (containing SBCS and DBCS data)
- Multi-byte character sets (MBCS)
- By default, to binary data in INTEL format.

No conversion

This applies to:

- Character data encoded as UCS-2 or UTF-8
- By default, to binary data in z/Architecture® format
- Packed decimal data.

User-defined nonstandard conversion

You can apply nonstandard data conversion by writing your own version of the user-replaceable conversion program.

You can apply user-defined conversion to selected fields, and leave others to be converted by the CICS standard conversion program.

For CICS Transaction Server for z/OS, you can provide *either*:

1. Your own, customized, version of DFHUCNV, *or*
2. One or more differently-named conversion programs

If the nonstandard conversion applies only to character data, you may not need to write your own data conversion program. Instead, you can create your own conversion tables for use with the standard conversion program, DFHCCNV. See "User-defined conversion tables" on page 238.

⚠ **Attention:** Your user-supplied conversion program must not convert any data that the standard conversion program attempts to convert. Converting data twice gives unpredictable results. To avoid this, your conversion program must convert only fields defined as DATATYP=USERDATA (see the DATATYP option of the DFHCNV TYPE=FIELD macro).

Character data

Character data is described by a *character set identifier* and a *code page identifier*. The code page identifier defines how each character is to be encoded; for example "A" is encoded as X'41' in ASCII and as X'C1' in EBCDIC.

The SRVERCP keyword on the DFHCNV TYPE=ENTRY macro specifies the EBCDIC code page in which character data associated with a resource is encoded in CICS Transaction Server for z/OS.

The CLINTCP keyword on the DFHCNV TYPE=ENTRY macro specifies the default code page in which the character data associated the specified resource is encoded when it is received by or sent from the CICS Transaction Server for z/OS. Typically, the data is encoded in ASCII, although in some cases it might be encoded in EBCDIC. When the data is encoded in EBCDIC, the code page is likely to be different from that specified by the SRVERCP keyword.

The code page specified by the CLINTCP keyword can be overridden. This allows CICS Transaction Server for z/OS to communicate with several systems, each of which uses a code page to represent character data.

Binary data

The DATATYP keyword on the DFHCNV TYPE=ENTRY macro specifies the default format for binary data received by CICS Transaction Server for z/OS.

DATATYP=BINARY
: Specifies that the default format for binary data is big-endian; that is, multibyte numerical values have the most significant byte values first (in the lower machine address).

DATATYP=NUMERIC
: Specifies that the default format for binary data is little-endian; that is, multibyte numerical values have the least significant byte values first.

The default binary format can be overridden. It is therefore important that you code a DFHCNV TYPE=FIELD macro for every binary field.

Resource definition to enable data conversion

To convert data in CICS Transaction Server for z/OS, you must define some resources in your CICS region.

You must define the following resources:

- DFHCNV, conversion table.

  You might have different versions of the DFHCNV table, for example, to deal with different versions of applications, or production and test systems. One way to ensure that the appropriate DFHCNV*xx* load modules are used is to put the DFHCNV program into a LIBRARY, so that the group with this library is installed in the region with the appropriate level of programs.
- DFHCCNV, standard conversion program.
- DFHUCNV, user-defined conversion program.

Defining the conversion table

You define the conversion table with DFHCNV resource definition macros.

The output of the DFHCNV macro assembly contains templates specifying resource conversion requirements and conversion tables to enable the required conversions. User-generated conversion tables must be placed in the DFHCNV macro source.

DFHCNV macro types

Use the DFHCNV macro to define the conversion table.

DFHCNV TYPE=INITIAL
: Defines the beginning of the conversion table. It defines the default client and server CCSIDs.

DFHCNV TYPE=ENTRY
: Specifies a name and type to uniquely identify a data resource. Specify a DFHCNV TYPE=ENTRY macro for each resource for which conversion is required; data is not converted for resources that are not defined in a DFHCNV TYPE=ENTRY macro. The entry for one resource is concluded by the next TYPE=ENTRY statement, or by the end of the table. The CCSID to be used is specified.

  You can create generic templates that apply to multiple resources of the same resource type. You do this by using the RPFX or XRPFX parameters of the DFHCNV TYPE=ENTRY macro to specify a prefix that can be matched against multiple resource names, rather than using the full name of a specific resource.

  Defining resources in this way means that sequence is important in the conversion table. For example, when specifying file resources, if prefix AB precedes prefix ABCD, the former entry is used to convert data for a file resource named ABCDEFGH. This example would give you an error when assembling the conversion table. To avoid errors, you should put the most specific resource names at the top of the conversion table, with the least specific prefix at the bottom.

When no resource name or prefix is specified, the default conversion template is used for that particular resource type.

For an example of the DFHCNV TYPE=ENTRY macro, see "DFHCNV TYPE=ENTRY" on page 233.

DFHCNV TYPE=KEY
Applies only to an FC entry. Use this macro only if a record might need to be accessed by key (if records are always accessed by relative record number or relative byte address, do not code a TYPE=KEY macro). If you use this macro, it must immediately follow a TYPE=ENTRY macro, and must be followed by one or more TYPE=FIELD macros, which define the data conversion to be applied to the key.

DFHCNV TYPE=SELECT
Defines selection of a record (FC record, TS data, TD data, IC start "from" data, or COMMAREA transmitted with DPL) for data conversion based on the value of a field in the record. Each TYPE=SELECT macro is followed by one or more TYPE=FIELD macros, which define the data conversion to be applied if the record satisfies the test defined in the TYPE=SELECT macro. The last TYPE=SELECT macro for each entry is an OPTION=DEFAULT macro, which defines the conversion to be applied to a record that satisfies no preceding TYPE=SELECT macro.

DFHCNV TYPE=FIELD
Specifies the position and length of a field, and the conversion to be applied to it. You must specify a TYPE=FIELD macro for each field for which conversion is required.

DFHCNV TYPE=FINAL
Concludes the conversion table definition.

Conversion and key templates

Templates are table entries defining fields in a data area or key that are to be converted and the conversion method to be applied to each field. There are two types of template: *conversion templates* and *key templates*.

- A conversion template is defined by one or more DFHCNV TYPE=FIELD macros following a DFHCNV TYPE=SELECT macro.
- A key template is defined by of one or more DFHCNV TYPE=FIELD macros following a DFHCNV TYPE=KEY macro.

Both types of template are terminated by the next non-FIELD macro in the table definition. Figure 62 on page 230 shows templates within a complete conversion table definition.

Defaults for client and server code pages

In order to reduce the number of conversion tables required, you can specify that the default client or server code page is defined in the system initialization table.

For the client code page:

1. In the DFHCNV TYPE=ENTRY and TYPE=SELECT macros, specify the value SYSDEF for the CLINTCP parameter.
2. In the system initialization table, set a default client code page by specifying a value for the CLINTCP parameter. You can use any value supported for the CLINTCP parameter on the DFHCNV macro. The default is CLINTCP=437.

For the server code page:

1. In the DFHCNV TYPE=ENTRY and TYPE=SELECT macros, specify the value SYSDEF for the SRVERCP parameter.
2. In the system initialization table, set a server code page by specifying a value for the SRVERCP parameter. You can use any value supported for SRVERCP parameter on the DFHCNV macro. The default is SRVERCP=037.

Conversion table for initial program verification (IVP)

When running the IVP jobs for CICS Transaction Server for z/OS, you need a conversion table.

Figure 61 on page 229 is a simple example of a conversion table definition. You don't need to code all these macros. You can generate exactly the same conversion table by assembling the special macro, DFHCNV TYPE=IVP.

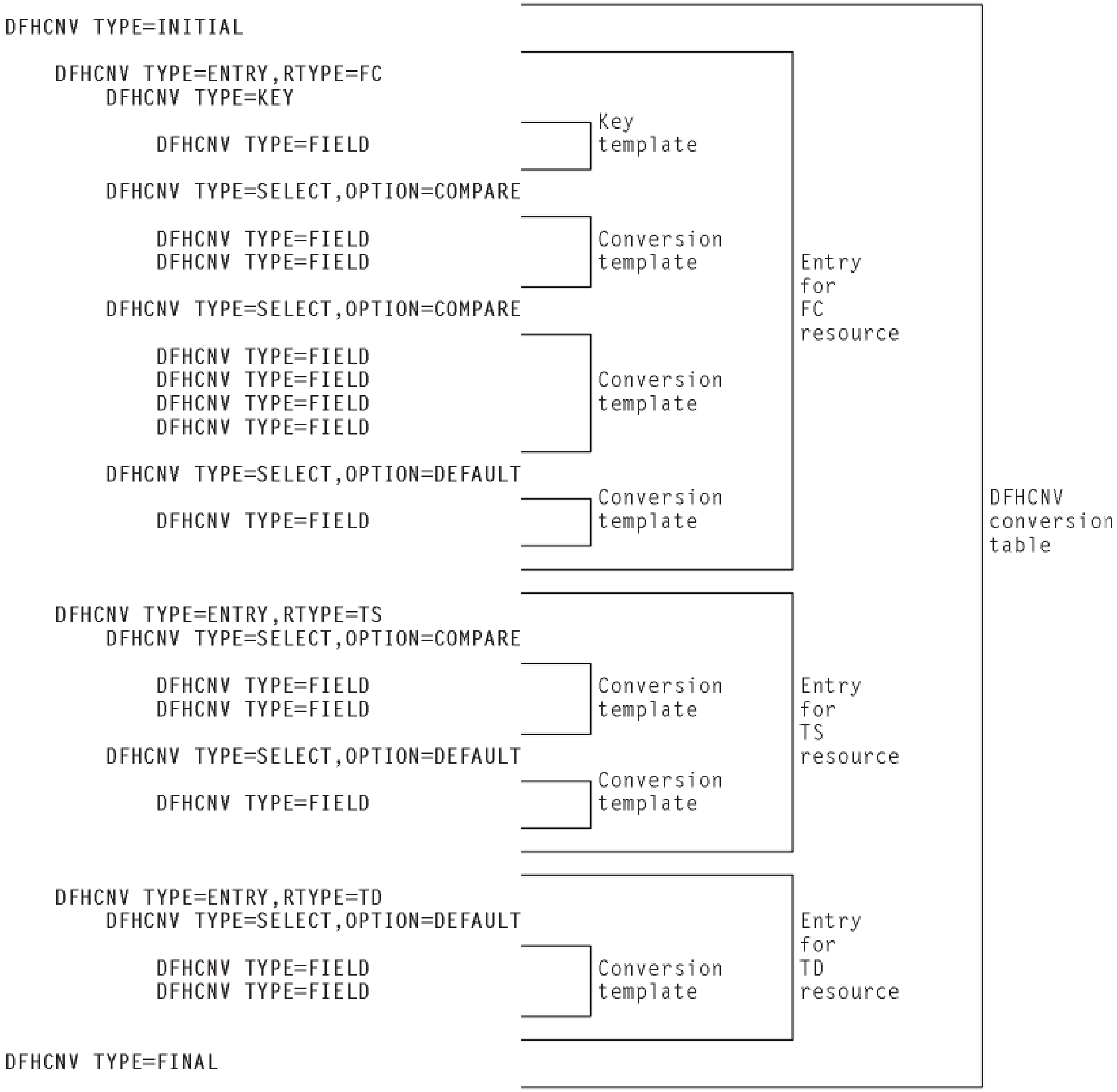
All the fields are character, so only a single TYPE=SELECT macro is needed. It specifies OPTION=DEFAULT, and has a single TYPE=FIELD macro to define the whole data record.

The TYPE=KEY macro is followed by a single TYPE=FIELD macro, which redefines the first six bytes of the data record.

```
    DFHCNV TYPE=INITIAL
    DFHCNV TYPE=ENTRY,RTYPE=FC,RNAME=FILEA,USREXIT=NO
    DFHCNV TYPE=KEY
    DFHCNV TYPE=FIELD,OFFSET=0,DATATYP=CHARACTER,DATALEN=6,LAST=YES
    DFHCNV TYPE=SELECT,OPTION=DEFAULT
    DFHCNV TYPE=FIELD,OFFSET=0,DATATYP=CHARACTER,DATALEN=80,LAST=YES
    DFHCNV TYPE=FINAL
```

Figure 61. Conversion table for IVP

Figure 62 on page 230 shows a typical sequence of DFHCNV macros. The figure is annotated to show the sets of entries that correspond to resource entries, conversion templates, and key templates. (The indentation is to illustrate nesting. When coding the macros, as with all CICS resource definition macros, observe assembler rules.)

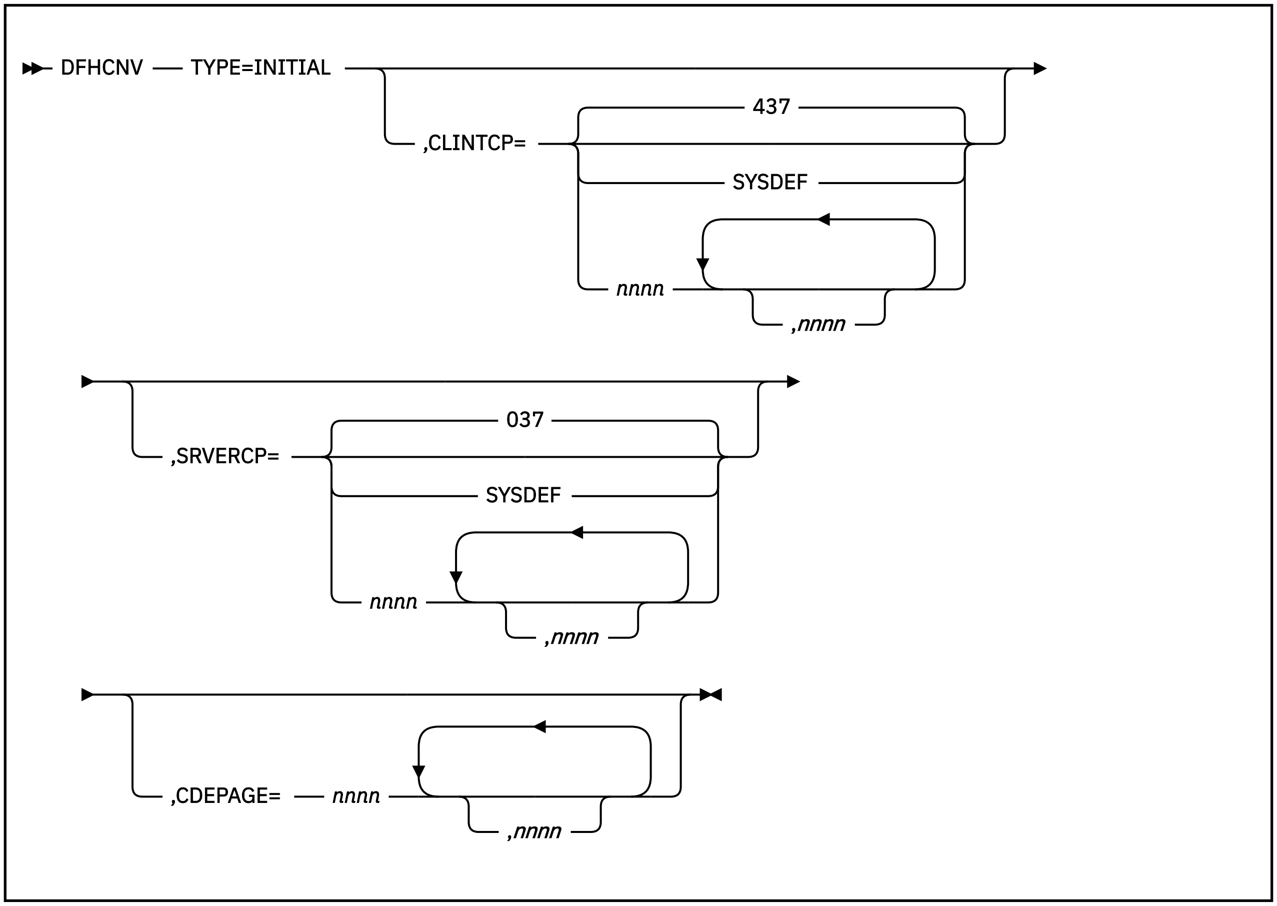


```
DFHCNV TYPE=INITIAL

    DFHCNV TYPE=ENTRY,RTYPE=FC
        DFHCNV TYPE=KEY

            DFHCNV TYPE=FIELD

        DFHCNV TYPE=SELECT,OPTION=COMPARE

            DFHCNV TYPE=FIELD
            DFHCNV TYPE=FIELD

        DFHCNV TYPE=SELECT,OPTION=COMPARE

            DFHCNV TYPE=FIELD
            DFHCNV TYPE=FIELD
            DFHCNV TYPE=FIELD
            DFHCNV TYPE=FIELD

        DFHCNV TYPE=SELECT,OPTION=DEFAULT

            DFHCNV TYPE=FIELD


    DFHCNV TYPE=ENTRY,RTYPE=TS
        DFHCNV TYPE=SELECT,OPTION=COMPARE

            DFHCNV TYPE=FIELD
            DFHCNV TYPE=FIELD

        DFHCNV TYPE=SELECT,OPTION=DEFAULT

            DFHCNV TYPE=FIELD


    DFHCNV TYPE=ENTRY,RTYPE=TD
        DFHCNV TYPE=SELECT,OPTION=DEFAULT

            DFHCNV TYPE=FIELD
            DFHCNV TYPE=FIELD

DFHCNV TYPE=FINAL
```

Figure 62. Example of DFHCNV macro sequence

DFHCNV TYPE=INITIAL

This is the format of the DFHCNV TYPE=INITIAL macro.



TYPE=INITIAL

Defines the beginning of the conversion table.

CLINTCP={437|SYSDEF|nnnn[,nnnn, ...]}

The first operand defines the default client CCSID to be used when the CLINTCP and CDEPAGE operands are omitted from a DFHCNV TYPE=ENTRY macro.

SYSDEF specifies that the default client code page is determined by the system initialization table parameter CLINTCP.

For an explanation of code pages, and a list of those that you can specify, see "Character data" on page 226.

SRVERCP={037|SYSDEF|nnnn[,nnnn, ...]}

The first operand defines the server CCSID to be used when the SRVERCP and CDEPAGE operands are omitted from a DFHCNV TYPE=ENTRY macro.

SYSDEF specifies that the default server code page is determined by the system initialization table parameter SRVERCP.

For an explanation of code pages, and a list of those that you can specify, see "Character data" on page 226.

CDEPAGE=nnnn[,nnnn...]

Restriction: Do not use this parameter for new definitions. It is supported only for compatibility with earlier releases.

Each possible value is equivalent to a pair of CLINTCP and SRVERCP entries or (for user-defined conversion) to a SRVERCP entry.

437
Is equivalent to:

- CLINTCP=437
- SRVERCP=037

932K
Is equivalent to:

- CLINTCP=932
- SRVERCP=930

932
Is equivalent to:

- CLINTCP=932
- SRVERCP=931

USR
Is equivalent to:

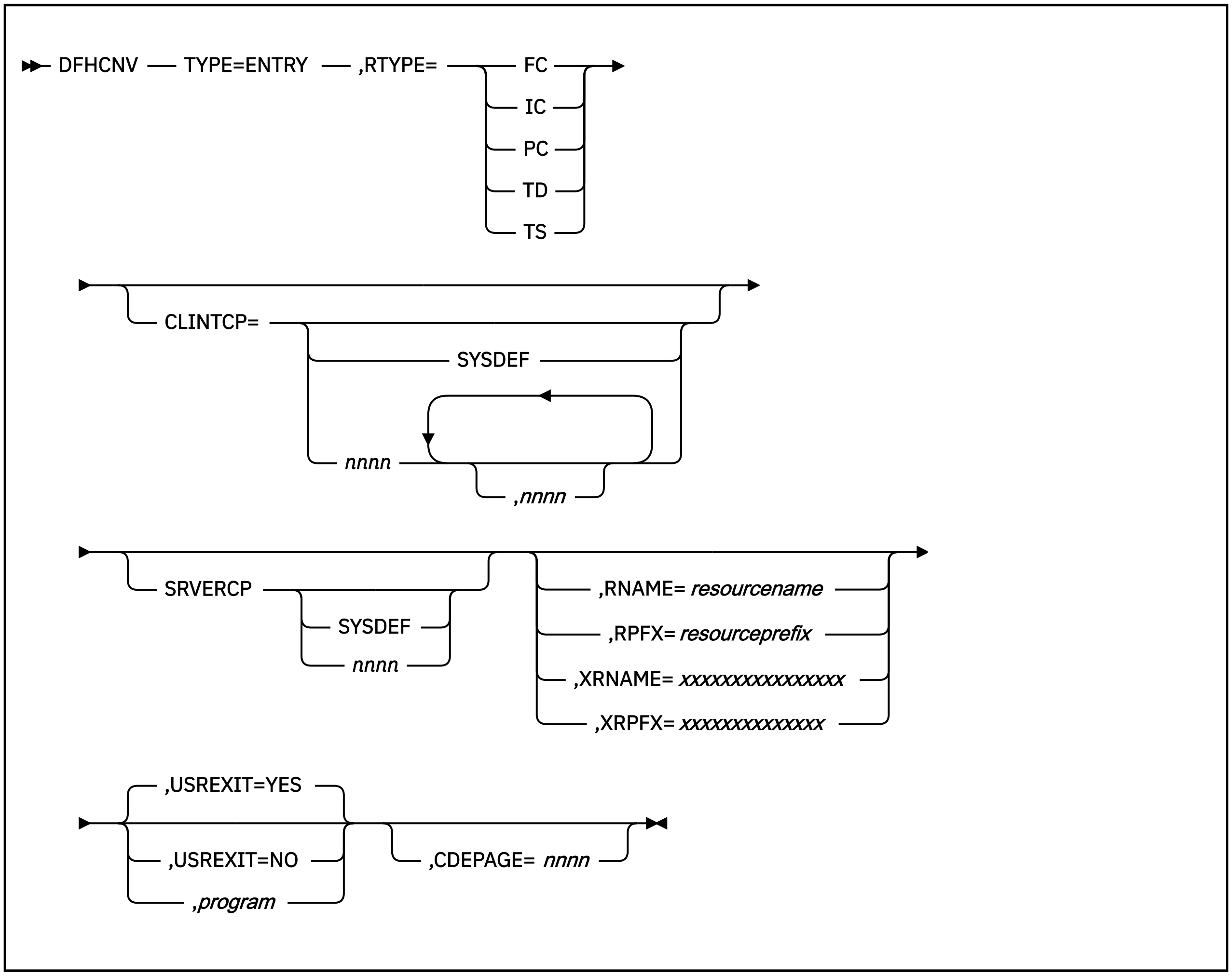
- SRVERCP=USR

USRD
Is equivalent to:

- SRVERCP=USRD

DFHCNV TYPE=ENTRY

This is the format of the DFHCNV TYPE=ENTRY macro instruction.



TYPE=ENTRY
: Specifies that this macro defines a resource by name and type.

RTYPE={FC|TS|TD|IC|PC}
: Specifies the type of resource:

FC
: A file

TS
: A temporary storage queue

TD
: A transient data queue

IC
: An interval control start with data

PC
: A program link with a COMMAREA.

CLINTCP={nnnn[,nnnn, ...]|SYSDEF}
: The first operand defines the default client code page to be used.

SYSDEF specifies that the default client code page is determined by the system initialization table parameter CLINTCP.

For an explanation of code pages, and a list of those that you can specify, see “Character data” on page 226.

SRVERCP={nnnn|SYSDEF}

The operand defines the server code page to be used.

SYSDEF specifies that the server code page is determined by the system initialization table parameter SRVERCP.

For an explanation of code pages, and a list of those that you can specify, see “Character data” on page 226.

RNAME=resourcename

Specifies the name of the resource in up to eight characters. If shorter, it is padded with blanks; if longer, it is truncated. The name can be:

- A FILE name (up to eight characters).
- A TS queue name (up to eight characters).

  Note: Although CICS supports TS queue names of up to16 characters, DFHCNV only supports TS queue names of up to 8 characters.
- A TD queue name (up to four characters).
- An IC start transaction id (up to four characters).
- The name of the program being linked (up to eight characters).

RPFX=resourceprefix

Specifies a resource prefix of up to 7 characters for programs, TS queues and files; or 3 characters for TD queues and transactions. The resource prefix allows resources of a particular type to be grouped together using just one macro. All resources of the specified type and prefix will be treated in the same way. Order is important, so the most specific resource names should be at the top of the conversion table, with the least specific prefixes at the bottom. If none of the parameters are specified at this point in the macro, the default template is used for all resources within the specified resource type.

XRNAME=xxxxxxxxxxxxxxxx (RTYPE=TS only)

Specifies the resource name in hexadecimal notation. It can include up to 16 hexadecimal digits, padded with blanks if necessary.

XRPFX=xxxxxxxxxxxxxx (RTYPE=TS only)

Specifies a resource prefix of up to 14 hexadecimal digits. The resource prefix allows resources of a particular type to be grouped together. All resources of the specified type and prefix will be treated in the same way. The sequence is important, so the most specific resource names should be at the top of the conversion table, with the least specific prefixes at the bottom. If none of the parameters are specified at this point in the macro, the default template is used for all resources within the specified resource type.

USREXIT={YES|NOv*program*}

Specifies whether the user data conversion exit is called.

YES

User-defined conversion is required for this resource. DFHUCNV is invoked. Code this if you need your customized version of DFHUCNV to convert some data for this resource.

NO

User-defined conversion is not required for this resource. The user-replaceable conversion program is not called. Code this to eliminate the overhead of calling the program unnecessarily.

program

User-defined conversion is required for this resource; *program* is invoked. Code this if you need your user-supplied program, *program*, to convert some data for this resource.

CDEPAGE=nnnn

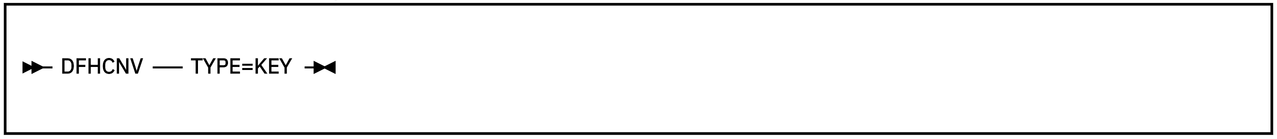
Restriction: Do not use this parameter for new definitions. It is supported only for compatibility with earlier releases.

The code page must be one of those entered in the CDEPAGE option of the DFHCNV TYPE=INITIAL macro. Each possible value is equivalent to a pair of CLINTCP and SRVERCP entries or (for user-defined conversion) to a SRVERCP entry. The CLINTCP and SRVERCP values to which each value resolves are given in the description of the CDEPAGE option of the DFHCNV TYPE=INITIAL macro.

DFHCNV TYPE=KEY

The DFHCNV TYPE=KEY macro is valid only for FC RTYPE requests, and, if coded, must immediately follow a DFHCNV TYPE=ENTRY macro.

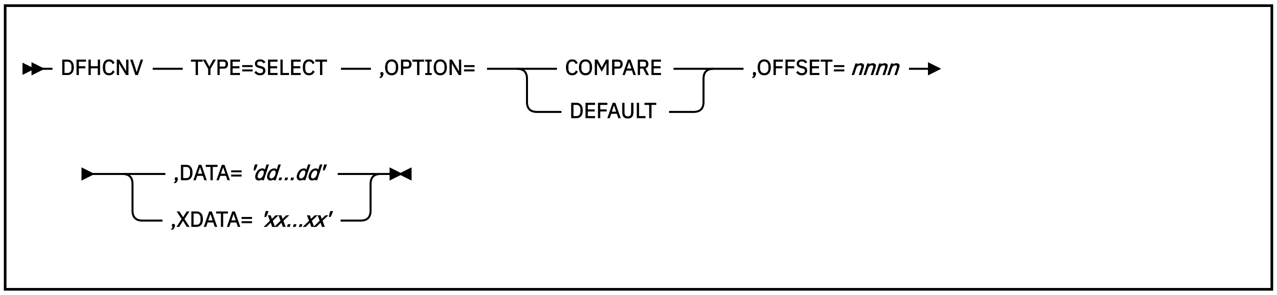
The macro has the following format:



TYPE=KEY
Indicates the start of conversions to be applied to a key. This macro is not required if access is only by RRN or RBA. If access is by key but no TYPE=KEY statement is present, the key is not converted. You must provide matching conversion details (DFHCNV TYPE=FIELD macros) for the key, as part of each conversion template that applies to this file, or an INVREQ condition may be returned on the file control EXEC CICS request.

DFHCNV TYPE=SELECT

This is the format of the DFHCNV TYPE=SELECT macro instruction.



TYPE=SELECT
Indicates the start of conversion definitions (DFHCNV TYPE=FIELD macros) to be applied to a record that satisfies the comparison defined in this macro. If the defined comparison is not satisfied by the data in the record, the conversion program (DFHCCNV) skips to the next TYPE=SELECT macro, until it finds a match or reaches the OPTION=DEFAULT macro. Every TYPE=SELECT macro must be followed by at least one TYPE=FIELD macro.

OPTION={COMPARE|DEFAULT}
States the basic selection options:

COMPARE
Indicates that the data should be converted according to the specifications in the following DFHCNV TYPE=FIELD macros, if the record satisfies the comparison defined in this macro (OFFSET and DATA or XDATA options).

DEFAULT
Indicates that the data should be converted according to the specifications in the following DFHCNV TYPE=FIELD macros, if the record has not satisfied the comparison defined in any previous DFHCNV TYPE=SELECT COMPARE macro.

For each resource entry (started by a TYPE=ENTRY macro) the last TYPE=SELECT macro must specify OPTION=DEFAULT. No other TYPE=SELECT macro in the entry should specify OPTION=DEFAULT.

The following options are ignored if OPTION=DEFAULT is coded.

OFFSET=nnnn

Specifies the byte offset in the record at which the comparison should be made, up to a maximum of 65535.

DATA='dd...dd'

Restriction: Use only if the data to be tested is defined as DATATYP=CHARACTER, SOSI=NO

Specifies the comparison data as an EBCDIC character string, with a maximum length of 255 characters. CICS converts the incoming data from ASCII to EBCDIC before checking it against the comparison data, so that EBCDIC is compared with EBCDIC. Outgoing data is in EBCDIC, so the comparison is made in EBCDIC without conversion.

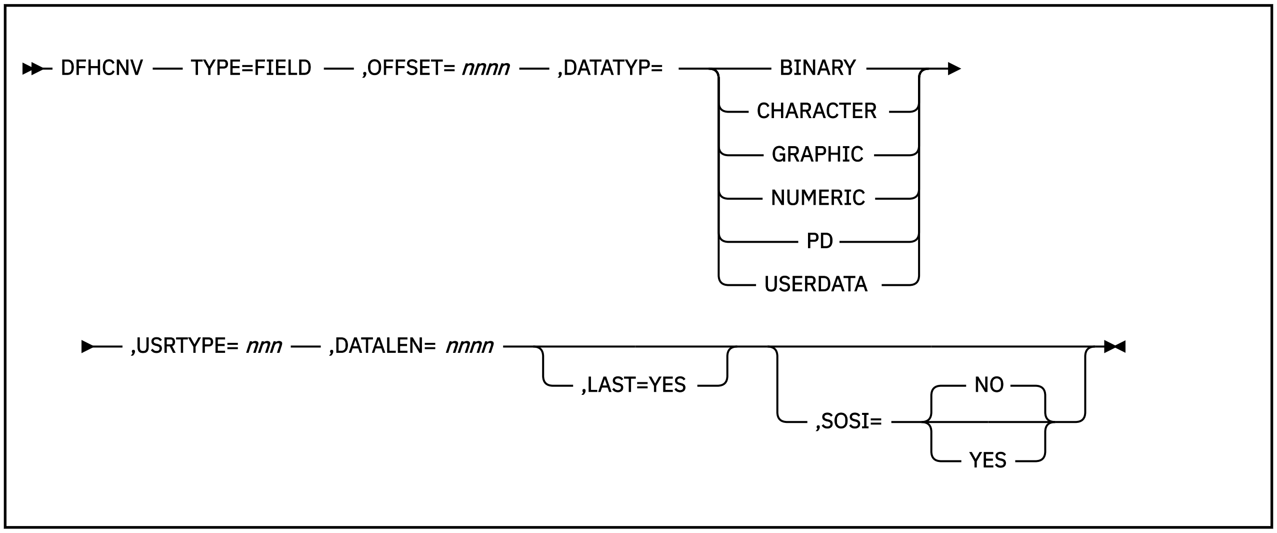
XDATA='xx...xx'

Restriction: Use if DATA option is not used

Specifies the comparison data as a hexadecimal string, with an even number of digits, maximum length 254 digits. Data is compared against this field, without conversion.

DFHCNV TYPE=FIELD

This is the format of the DFHCNV TYPE=FIELD macro instruction, which occurs as many times as needed.



TYPE=FIELD

Specifies conversion specifications for a data field. There must be one such statement for each field in a record. You cannot code a TYPE=FIELD macro until you have coded a TYPE=SELECT macro.

OFFSET=nnnn

Specifies the byte offset in the record or key at which the conversion should start, up to a maximum of 65535. (For TYPE=KEY conversions, this is the byte offset from the start of the *key* not from the start of the record.)

DATATYP={CHARACTER|PD|BINARY|USERDATA|GRAPHIC|NUMERIC}

Specifies the type of conversion required:

CHARACTER

Specifies character fields.

PD

Specifies packed decimal data in z/Architecture format.

Any packed decimal data in other formats should be defined for USERDATA conversion, and the user-replaceable program DFHUCNV must contain the necessary conversion code.

BINARY

Specifies binary data in big-endian format.

By default, BINARY data is not converted. This default action can be overridden to allow requests from platforms that support different binary architectures to access the same CICS resource using the same conversion table.

USERDATA

Specifies data to be converted by the user-replaceable program DFHUCNV. The DFHCCNV conversion code bypasses these fields. See the USRTYPE operand.

GRAPHIC

Specifies fields that contain DBCS characters only.

NUMERIC

Specifies that binary fields held on the workstation in INTEL format (for example, C Language integer datatype) need to be converted to z/Architecture format. Integers (four bytes) or short integers (two bytes) can be converted.

USRTYPE=nnn

Specifies a value that is made available to the user-replaceable conversion program DFHUCNV. The values you provide can be in the range 80 to 128 (X'50' to X'80'). The default value is 80 (X'50'). If more than one type of user-defined conversion is possible, you can use this value to specify to DFHUCNV what conversion is needed for each field.

This option is ignored if DATATYP=USERDATA is not specified.

DATALEN=n

Specifies the length of the data field to be converted, in bytes, up to a maximum of 65535. For variable length fields, specify the maximum possible length.

If DATATYP=NUMERIC, DATALEN must be 2 or 4.

LAST=YES

Specifies that this is the last field definition for this TYPE=SELECT statement.

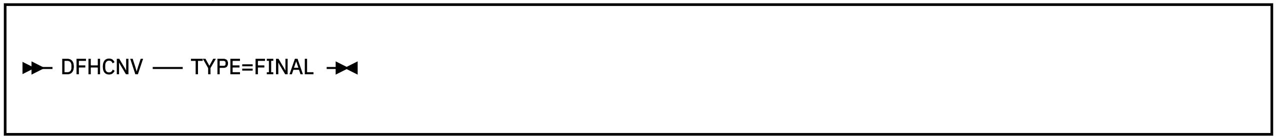
SOSI=YES|NO

Enter YES for a mixed string containing SBCS and DBCS characters; enter NO for an SBCS string. This field is valid only if DATATYPE=CHARACTER has been entered in this macro. The default is NO.

DFHCNV TYPE=FINAL

The DFHCNV TYPE=FINAL macro instruction ends the table.

It must occur only once, as the last definition.



Hints on coding the macros

You can improve the performance of data conversion by coding your macros to benefit from the way in which CICS processes the conversion tables.

1. Define entries for the most frequently-used resources first, to reduce search time.
2. Define USERDATA fields in consecutive entries. This reduces the time needed by your conversion program to scan the template.
3. For variable-length fields, define the maximum length required. (Comparisons and conversions are applied to the shorter of the actual data length or the template length. For example, if the data is 100 bytes long but the template describes 120 bytes, up to 100 bytes are converted. If the data is 100 bytes and the template describes 80 bytes, only 80 bytes are converted.)
4. If function-shipped data is not accessed by CICS Transaction Server for z/OS but only by the connected system, you do not need to specify conversion details. For example, when a CICS Transaction Server for z/OS file is used to store data that is shared by several ASCII-based systems.

User-defined conversion tables

If you specify SRVERCP=USR or USRD in a DFHCNV TYPE=ENTRY macro, you must provide user-defined conversion tables. The standard conversion program (DFHCCNV) uses these tables, and they are made available to the user-replaceable conversion program, DFHUCNV.

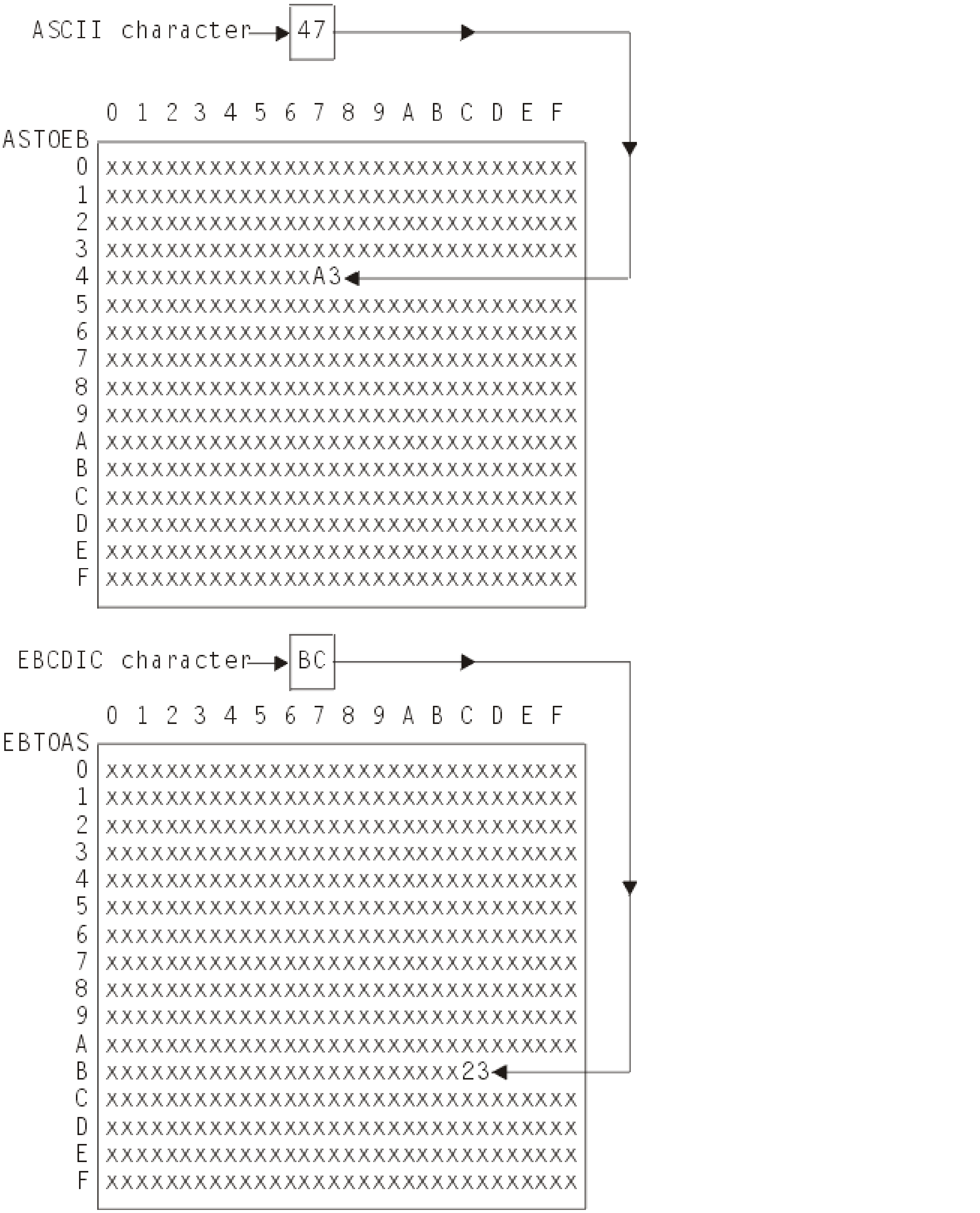
Place your user-defined conversion tables in the DFHCNV macro source, anywhere after the DFHCNV TYPE=INITIAL macro.

Tip: For source readability, the best place is probably after the DFHCNV TYPE=FINAL macro.

SRVERCP=USR

You must provide two character conversion tables, labelled ASTOEB and EBTOAS.

Each table must be 256 bytes long. ASTOEB is used for ASCII to EBCDIC conversion and EBTOAS is used for EBCDIC to ASCII conversion. The hexadecimal value of a character byte is used as an offset in the conversion table to obtain the converted value of the character. Figure 63 on page 239 illustrates this process.



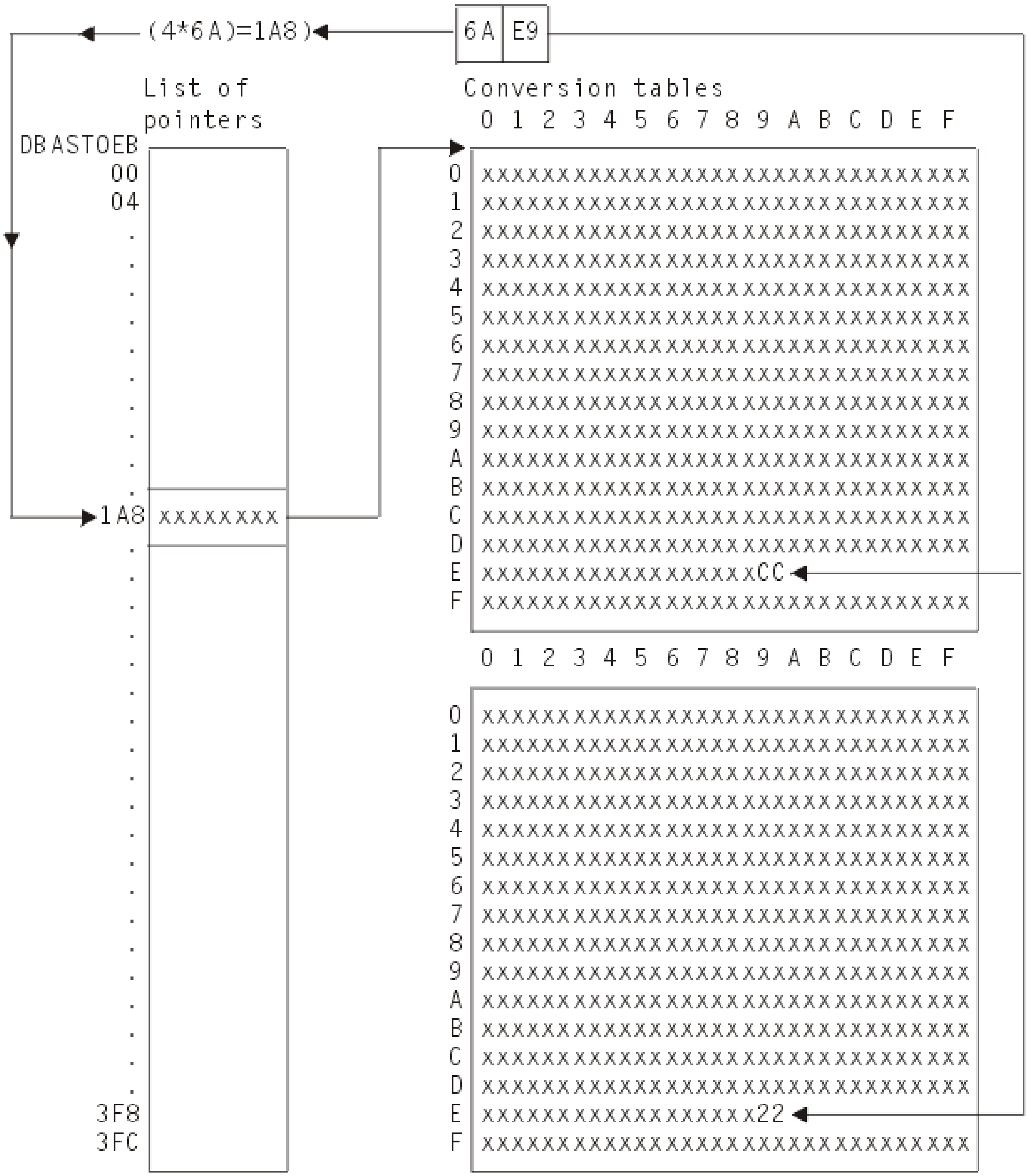
In this example, the ASCII character X'47' converts to the EBCDIC character X'A3', and the EBCDIC character X'BC' converts to the ASCII character X'23'.
These values have no significance, and are used simply to illustrate the structure of the conversion tables.

Figure 63. Structure of SBCS conversion tables

SRVERCP=USRD

You must provide DBCS character conversion tables labelled DBASTOEB and DBEBTOAS, in the DFHCNV source. These must be after the DFHCNV TYPE=INITIAL macro, but otherwise anywhere in the source. Each table must be a list of 256 four-byte pointers and 256 pairs of 256-byte translate tables. The first byte of a DBCS character is used as an index to the list of pointers. Using the first byte of the DBCS character as a hexadecimal offset in the list, the pointer found is the address of a pair of 256-byte translate tables. The second byte of the DBCS character is used as an offset in each of the two 256-byte translate tables to obtain the first and second bytes of the converted DBCS character. Figure 64 on page 241 illustrates this process.

You must also provide an SBCS conversion table as specified previously under USR.



In this example, the double-byte character X'6AE9' converts to X'CC22'. The value, at offset 6A in the pointer list, is the address of a pair of 256-byte translate tables. At offset E9 in these tables, the byte values are X'CC' and X'22' respectively. These are random values, used purely for illustration.

This is an ASCII-EBCDIC conversion, because the pointer list is labeled DBASTOEB. A complete set of ASCII-EBCDIC tables contains 256 pairs of 256-byte tables, one pair for each possible value of the first byte of a double-byte character.

DBEBTOAS is the label of a similar set of EBCDIC-ASCII tables.

Figure 64. Structure of DEBUTS conversion tables

Invalid and undefined DBCS characters

In ASCII and EBCDIC, certain code ranges are valid DBCS code. Any double-byte value outside these ranges is an invalid DBCS character. In the supplied conversion tables, invalid DBCS characters convert to X'FFFF', as defined by the code page architecture.

Within the valid code range, several thousand double-byte values are defined as actual DBCS characters. A double-byte value within the valid code range, but not defined as a DBCS character, is an undefined DBCS character.

User-defined tables should follow these conventions for invalid and undefined characters.

Example macros

These examples show the use of the data conversion macros.

Figure 65 on page 242 shows an example of a record layout for a file called VSAM99. The key is offset 0 for length 6, and the record contains no redefinition.

```
02  FILEREC.
  03  STAT         PIC X.
  03  NUMB         PIC X(6).
  03  NAME         PIC X(20).
  03  ADDRX        PIC X(20).
  03  PHONE        PIC X(8).
  03  DATEX        PIC X(8).
  03  AMOUNT       PIC X(8).
  03  COMMENT      PIC X(9).
  03  COUNTER1     PIC 9999 USAGE COMP-4.
  03  COUNTER2     PIC 9999 USAGE COMP-4.
  03  ADDLCMT      PIC X(30).
```

Figure 65. Record layout for VSAM99

Figure 66 on page 242 gives a full set of conversion macros for file VSAM99. Figure 67 on page 242 shows the same conversion expressed more briefly, by combining adjoining fields of the same type.

```
    DFHCNV TYPE=INITIAL,CLINTCP=437,SRVERCP=037
    DFHCNV TYPE=ENTRY,RTYPE=FC,RNAME=VSAM99
    DFHCNV TYPE=KEY
    DFHCNV TYPE=FIELD,OFFSET=0,DATATYP=CHARACTER,DATALEN=6,LAST=YES
    DFHCNV TYPE=SELECT,OPTION=DEFAULT
    DFHCNV TYPE=FIELD,OFFSET=00,DATATYP=CHARACTER,DATALEN=1
    DFHCNV TYPE=FIELD,OFFSET=01,DATATYP=CHARACTER,DATALEN=6
    DFHCNV TYPE=FIELD,OFFSET=07,DATATYP=CHARACTER,DATALEN=20
    DFHCNV TYPE=FIELD,OFFSET=27,DATATYP=CHARACTER,DATALEN=20
    DFHCNV TYPE=FIELD,OFFSET=47,DATATYP=CHARACTER,DATALEN=8
    DFHCNV TYPE=FIELD,OFFSET=55,DATATYP=CHARACTER,DATALEN=8
    DFHCNV TYPE=FIELD,OFFSET=63,DATATYP=CHARACTER,DATALEN=8
    DFHCNV TYPE=FIELD,OFFSET=71,DATATYP=CHARACTER,DATALEN=9
    DFHCNV TYPE=FIELD,OFFSET=80,DATATYP=BINARY,DATALEN=2
    DFHCNV TYPE=FIELD,OFFSET=82,DATATYP=BINARY,DATALEN=2
    DFHCNV TYPE=FIELD,OFFSET=84,DATATYP=CHARACTER,DATALEN=30,LAST=YES
    DFHCNV TYPE=FINAL
```

Figure 66. Full description of VSAM99

```
    DFHCNV TYPE=INITIAL,CLINTCP=437,SRVERCP=037
    DFHCNV TYPE=ENTRY,RTYPE=FC,RNAME=VSAM99
    DFHCNV TYPE=KEY
    DFHCNV TYPE=FIELD,OFFSET=0,DATATYP=CHARACTER,DATALEN=6,LAST=YES
    DFHCNV TYPE=SELECT,OPTION=DEFAULT
    DFHCNV TYPE=FIELD,OFFSET=00,DATATYP=CHARACTER,DATALEN=80
    DFHCNV TYPE=FIELD,OFFSET=80,DATATYP=BINARY,DATALEN=4
    DFHCNV TYPE=FIELD,OFFSET=84,DATATYP=CHARACTER,DATALEN=30,LAST=YES
    DFHCNV TYPE=FINAL
```

Figure 67. Condensed description of VSAM99

Note: Be careful when combining adjoining fields, even if they are of the same data type. Do not combine NUMERIC fields. Do not combine fields defined as CHARACTER, if SOSI=YES is specified for one or more of them. Whether you can combine USERDATA fields depends on user-defined data structures and conversion code.

Figure 68 on page 243 shows a redefined record layout for file VSAM99. Figure 69 on page 243 shows a set of conversion macros for the redefined record layout in Figure 68 on page 243.

```
02  FILEREC.
  03  STAT         PIC X.
  03  NUMB         PIC X(6).
  03  NAME         PIC X(20).
  03  ADDRX        PIC X(20).
  03  PHONE        PIC X(8).
  03  DATEX        PIC X(8).
  03  AMOUNT       PIC X(8).
  03  COMMENT      PIC X(9).
  03  VARINF1.
  03  COUNTER1     PIC 9999 USAGE COMP-4.
  03  COUNTER2     PIC 9999 USAGE COMP-4.
  03  ADDLCMT      PIC X(30).
  03  VARINF2 REDEFINES VARINF1.
  03  COUNTER1     PIC 9999 USAGE COMP-4.
  03  COUNTER2     PIC 9999 USAGE COMP-4.
  03  COUNTER3     PIC 9999 USAGE COMP-4.
  03  COUNTER4     PIC 9999 USAGE COMP-4.
  03  ADDLCMT2     PIC X(26).
```

Figure 68. Redefined record layout for VSAM99

```
   DFHCNV TYPE=INITIAL
   DFHCNV TYPE=ENTRY,RTYPE=FC,RNAME=VSAM99
   DFHCNV TYPE=KEY
   DFHCNV TYPE=FIELD,OFFSET=0,DATATYP=CHARACTER,DATALEN=6,LAST=YES
*
*  If offset 00 is a character 'X' use the following
*  conversion definitions:
*

   DFHCNV TYPE=SELECT,OPTION=COMPARE,OFFSET=00,DATA='X'
   DFHCNV TYPE=FIELD,OFFSET=00,DATATYP=CHARACTER,DATALEN=80
   DFHCNV TYPE=FIELD,OFFSET=80,DATATYP=BINARY,DATALEN=4
   DFHCNV TYPE=FIELD,OFFSET=84,DATATYP=CHARACTER,DATALEN=30,LAST=YES
*
*  Otherwise use the following (default)
*  conversion definitions
*
   DFHCNV TYPE=SELECT,OPTION=DEFAULT
   DFHCNV TYPE=FIELD,OFFSET=00,DATATYP=CHARACTER,DATALEN=80
   DFHCNV TYPE=FIELD,OFFSET=80,DATATYP=BINARY,DATALEN=8
   DFHCNV TYPE=FIELD,OFFSET=88,DATATYP=CHARACTER,DATALEN=26,LAST=YES
   DFHCNV TYPE=FINAL
```

Figure 69. Description for redefined record layout for VSAM99

Figure 70 on page 244 shows user-defined conversion tables, EBTOAS and ASTOEB, illustrating how they are preceded with DFHCNV macros in the source that is submitted to the assembler.

```
*
LABL1    DFHCNV TYPE=INITIAL,CLINTCP=437,SRVERCP=037
*
         DFHCNV TYPE=ENTRY,RTYPE=FC,RNAME=VSAM80
         DFHCNV TYPE=KEY
         DFHCNV TYPE=FIELD,OFFSET=0,DATATYP=BINARY,DATALEN=2
         DFHCNV TYPE=FIELD,OFFSET=2,DATATYP=CHARACTER,DATALEN=4,   X
               LAST=YES
LABLX    DFHCNV TYPE=SELECT,OPTION=COMPARE,OFFSET=6,XDATA='C1C2C3'
         DFHCNV TYPE=FIELD,OFFSET=0,DATATYP=BINARY,DATALEN=2
         DFHCNV TYPE=FIELD,OFFSET=2,DATATYP=CHARACTER,DATALEN=4
         DFHCNV TYPE=FIELD,OFFSET=9,DATATYP=CHARACTER,DATALEN=8,   X
               LAST=YES
                          ⋮
         DFHCNV TYPE=ENTRY,RTYPE=TS,RNAME=ABCD
         DFHCNV TYPE=SELECT,OPTION=DEFAULT
         DFHCNV TYPE=FIELD,OFFSET=0,DATATYP=CHARACTER,DATALEN=40
         DFHCNV TYPE=FIELD,OFFSET=40,DATATYP=BINARY,DATALEN=4,     X
               LAST=YES
LABLN    DFHCNV TYPE=FINAL
*
*  EXAMPLE OF A USER-DEFINED CONVERSION TABLE EBCDIC to ASCII
EBTOAS   DC    XL16'000102030405060708090A0B0C0D0E0F'
         DC    XL16'101112131415161718191A1B1C1D1E1F'
         DC    XL16'202122232425262728292A2B2C2D2E2F'
         DC    XL16'303132333435363738393A3B3C3D3E3F'
         DC    XL16'404142434445464748494A4B4C4D4E4F'
         DC    XL16'505152535455565758595A5B5C5D5E5F'
         DC    XL16'606162636465666768696A6B6C6D6E6F'
         DC    XL16'707172737475767778797A7B7C7D7E7F'
         DC    XL16'80C1C2C3C4C5C6C7C8C98A8B8C8D8E8F'
         DC    XL16'90D1D2D3D4D5D6D7D8D99A9B9C9D9E9F'
         DC    XL16'A0A1E2E3E4E5E6E7E8E9AAABACADAEAF'
         DC    XL16'B0B1B2B3B4B5B6B7B8B9BABBBCBDBEBF'
         DC    XL16'C0C1C2C3C4C5C6C7C8C9CACBCCCDCECF'
         DC    XL16'D0D1D2D3D4D5D6D7D8D9DADBDCDDDEDF'
         DC    XL16'E0E1E2A3E4E5E6E7E8E9EAEBECEDEEEF'
         DC    XL16'F0F1F2F3F4F5F6F7F8F9FAFBFCFDFEFF'
*
*  EXAMPLE OF A USER-DEFINED CONVERSION TABLE ASCII to EBCDIC
*
ASTOEB   DC    XL16'000102030405060708090A0B0C0D0E0F'
         DC    XL16'101112131415161718191A1B1C1D1E1F'
         DC    XL16'202122232425262728292A2B2C2D2E2F'
         DC    XL16'303132333435363738393A3B3C3D3E3F'
         DC    XL16'404142434445464748494A4B4C4D4E4F'
         DC    XL16'505152535455565758595A5B5C5D5E5F'
         DC    XL16'606162636465666768696A6B6C6D6E6F'
         DC    XL16'707172737475767778797A7B7C7D7E7F'
         DC    XL16'808182838485868788898A8B8C8D8E8F'
         DC    XL16'909192939495969798999A9B9C9D9E9F'
         DC    XL16'A0A1A2A3A4A5A6A7A8A9AAABACADAEAF'
         DC    XL16'B0B1B2B3B4B5B6B7B8B9BABBBCBDBEBF'
         DC    XL16'C08182838485868788899CACBCCCDCECF'
         DC    XL16'D09192939495969798999DADBDCDDDEDF'
         DC    XL16'E0E1A2A3A4A5A6A7A8A9EAEBECEDEEEF'
         DC    XL16'F0F1F2F3F4F5F6F7F8F9FAFBFCFDFEFF'
         END   DFHCNVBA
```

Figure 70. SBCS user-defined conversion table

Assembling and link-editing the conversion programs

You can use either of the standard procedures DFHAUPLE and DFHAUPLK to assemble the DFHCNV table.

About this task

You can optimize CICS virtual storage use by link-editing the DFHCNV table and the DFHUCNV program with a MODE statement specifying AMODE(31) and RMODE(ANY). The table and program are then loaded in 31-bit storage (above 16 MB but below 2 GB) if enough CICS storage is available.

Chapter 3. The user-replaceable conversion program

This section describes the user-replaceable data conversion program.

User-named conversion programs

You can replace DFHUCNV, the default user-replaceable conversion program, by one or more user-named conversion programs.

DFHUCNV is invoked if:

- A conversion template is not defined for the resource, *or*
- A conversion template is defined for the resource and the template specifies USREXIT=YES.

A user-named conversion program is invoked if:

- A conversion template is defined for the resource and the template specifies USREXIT=*userprogram*

  where *userprogram* is the name of the user-supplied conversion program.

Input to DFHUCNV

The first statement in the supplied version of DFHUCNV is a DFHCNV TYPE=DSECT macro, which generates DSECTs that describe the parameter list and the conversion template.

DFHUCNV starts with a DFHCNV TYPE=DSECT in the following format:

```
DFHCNV   TYPE=DSECT
```

The DFHCNV TYPE=DSECT macro generates the following:

- The DFHUNVDS DSECT, which maps the parameter list in the COMMAREA passed by DFHCCNV.
- An assembler DSECT for field conversion records (these are the basic components of a template; see Figure 73 on page 249).
- Equates for resource types and field types.

Parameter list (DFHUVNDS)

The DFHUNVDS DSECT maps the parameter list passed to DFHUCNV in the COMMAREA.

If a parameter is zero, no data is available. *If you do not create a conversion template for the resource, DFHUCNV is invoked, but only the following fields in the parameter list contain data*:

- UNVRSTP
- UNVRNMP
- UNVDIRP
- UNVOVLY

```
DFHUNVDS DSECT
UNVRSTP  DS    AL4                 PTR-TO-RESOURCE TYPE
UNVRNMP  DS    AL4                 PTR-TO-RESOURCE NAME
UNVDIRP  DS    AL4                 PTR-TO-CONVERSION DIRECTIVE
CNVRQATE EQU   X'02'               REQUEST ASCII TO EBCDIC
CNVRPETA EQU   X'04'               RESPONSE EBCDIC TO ASCII
UNVDTMP  DS    AL4                 PTR-TO-DATA CONV TEMPLATE
UNVDLNP  DS    AL4                 PTR-TO-DATA TEMPLATE LENGTH
UNVKTMP  DS    AL4                 PTR-TO-KEY CONV TEMPLATE
UNVKLNP  DS    AL4                 PTR-TO-KEY TEMPLATE LENGTH
UNVATEP  DS    AL4                 PTR-TO-ASCII/EBCDIC TRANS TABLE
UNVETAP  DS    AL4                 PTR-TO-EBCDIC/ASCII TRANS TABLE
UNVATED  DS    AL4                 PTR-TO-DBCS ASCII/EBCDIC TRANS TABLE
UNVETAD  DS    AL4                 PTR-TO-DBCS EBCDIC/ASCII TRANS TABLE

UNVOVLY  DS    0H                  OVERLAY SECTION
         ORG   UNVOVLY             TS REQUEST OVERLAY
UNVTSDP  DS    AL4                 PTR-TO-TS DATA
UNVTSLNP DS    AL4                 PTR-TO-TS DATA LENGTH
         ORG   UNVOVLY             TD REQUEST OVERLAY
UNVTDDP  DS    AL4                 PTR-TO-TD DATA
UNVTDLNP DS    AL4                 PTR-TO-TD DATA LENGTH
         ORG   UNVOVLY             IC REQUEST OVERLAY
UNVICDP  DS    AL4                 PTR-TO-IC DATA
UNVICLNP DS    AL4                 PTR-TO-IC DATA LENGTH
         ORG   UNVOVLY             PC REQUEST OVERLAY
UNVPCDP  DS    AL4                 PTR-TO-PC DATA
UNVPCLNP DS    AL4                 PTR-TO-PC DATA LENGTH
         ORG   UNVOVLY             FC REQUEST OVERLAY
UNVFCDP  DS    AL4                 PTR-TO-FC DATA
UNVFCLNP DS    AL4                 PTR-TO-FC DATA LENGTH
UNVFCKP  DS    AL4                 PTR-TO-FC KEY
UNVFCKLP DS    AL4                 PTR-TO-FC KEY  LENGTH
         ORG   ,
UNVMRTNE DS    A                   PTR-TO-MBCS TRANSLATION ROUTINE
UNVCLIDP DS    AL4                 A "client" CCSID
*                                    (for example, 00819)
UNVSRIDP DS    AL4                 A "server" CCSID
*                                    (for example, 00285)
```

Figure 71. DFHUNVDS—DSECT that maps the parameter list passed to DFHUCNV

The following is a detailed description of the parameters:

UNVRSTP
Points to a one-byte resource type that indicates the resource being referenced by this request. The meanings of the resource types are defined in DSECT DFHCNVDS. The resource types are FC, IC, TS, TD, and PC.

UNVRNMP
Points to an eight-character field containing the resource name, padded with blanks if necessary. These may be:

- For an FC request, an eight-byte file name
- For a TS request, an eight-byte TS queue name
- For a TD request, a four-byte TD queue name
- For an IC request, a four-byte transaction name
- For a PC request, an eight-byte program name.

UNVDIRP
Points to a one-byte field that shows what conversion is required:

- CNVRQATE (X'02') indicates a request needing conversion from client encoding to server encoding.
- CNVRPETA (X'04') indicates a response needing conversion from server encoding to client encoding.

UNVDTMP
Points to the start of the conversion template found by CICS to match this resource. If UNVDTMP is zero no template was found.

UNVDLNP
Points to a field that gives the length of the conversion template. The field is:

- A fullword for CICS Transaction Server for z/OS
- A half-word for all other platforms.

UNVKTMP (file control requests only)
Points to the start of the template found by CICS for the key part of the request or response. If UNVKTMP is zero, either there is no key template or the record is accessed by relative record number or relative byte address.

UNVKLNP (file control requests only)
Points to a field that gives the length of the key conversion template. The field is:

- A fullword for CICS Transaction Server for z/OS
- A half-word for all other platforms.

UNVATEP
Points to a 256-byte SBCS translation table used for converting character data from client encoding to server encoding.

UNVETAP
Points to a 256-byte SBCS translation table used for converting character data from server encoding to client encoding.

UNVATED
Points to a DBCS translation table used for converting character data from client encoding to server encoding.

UNVETAD
Points to a DBCS translation table used for converting character data from server encoding to client encoding.

The overlay section depends on resource type:

TS requests:

UNVTSDP
Points to the start of the TS record being read or written. The field is:

- A fullword for CICS Transaction Server for z/OS
- A half-word for all other platforms.

UNVTSLNP
Points to a field that gives the length of the TS record.

TD requests:

UNVTDDP
Points to the start of the TD record being read or written.

UNVTDLNP
Points to a field that gives the length of the TD record. The field is:

- A fullword for CICS Transaction Server for z/OS
- A half-word for all other platforms.

IC requests:

UNVICDP
Points to the “from” area of an IC START request.

UNVICLNP
Points to a field that gives the length of the “from” area. The field is:

- A fullword for CICS Transaction Server for z/OS
- A half-word for all other platforms.

PC requests:

UNVPCDP

Points to the start of the COMMAREA being supplied.

UNVPCLNP

Points to a field that gives the length of the COMMAREA. The field is:

- A fullword for CICS Transaction Server for z/OS
- A half-word for all other platforms.

FC requests:

UNVFCDP

Points to the start of the file control record being read or written.

UNVFCLNP

Points to a field that gives the length of the file control record. The field is:

- A fullword for CICS Transaction Server for z/OS
- A half-word for all other platforms.

UNVFCKP

Points to the start of the key for the file control record being read or written.

UNVFCKLP

Points to a field that gives the length of the key. The field is:

- A fullword for CICS Transaction Server for z/OS
- A half-word for all other platforms.

UNVMRTNE

Points to a translation routine that must be used for translations to or from an MBCS code page. The relevant client code pages are 954, 964, and 970.

The routine expects Register 1 to point to a structure defined by the DFHUNVM DSECT:

```
DFHUNVM DSECT
UNVMTABP  DS    AL4            Set to value in UNVATED or UNVETAD
UNVMINP   DS    AL4            Address of source data
INVMINL   DS    FL4            Length of source data
UNVMOUTP  DS    AL4            Address of target buffer
UNVMOUTL  DS    FL4            Length of target buffer
```

UNVCLIDP

Points to a fullword field that gives the IBM-defined CCSID, for example 00819, corresponding to the “client” code page.

UNVSRIDP

Points to a fullword field that gives the IBM-defined CCSID, for example 00285, corresponding to the “server” code page.

Conversion and key templates

In the COMMAREA, fields UNVDTMP and UNVDLNP point to the conversion template and its length.

Fields UNVKTMP and UNVKLNP point to the key template and its length. Figure 72 on page 249 illustrates the use and meaning of these fields.

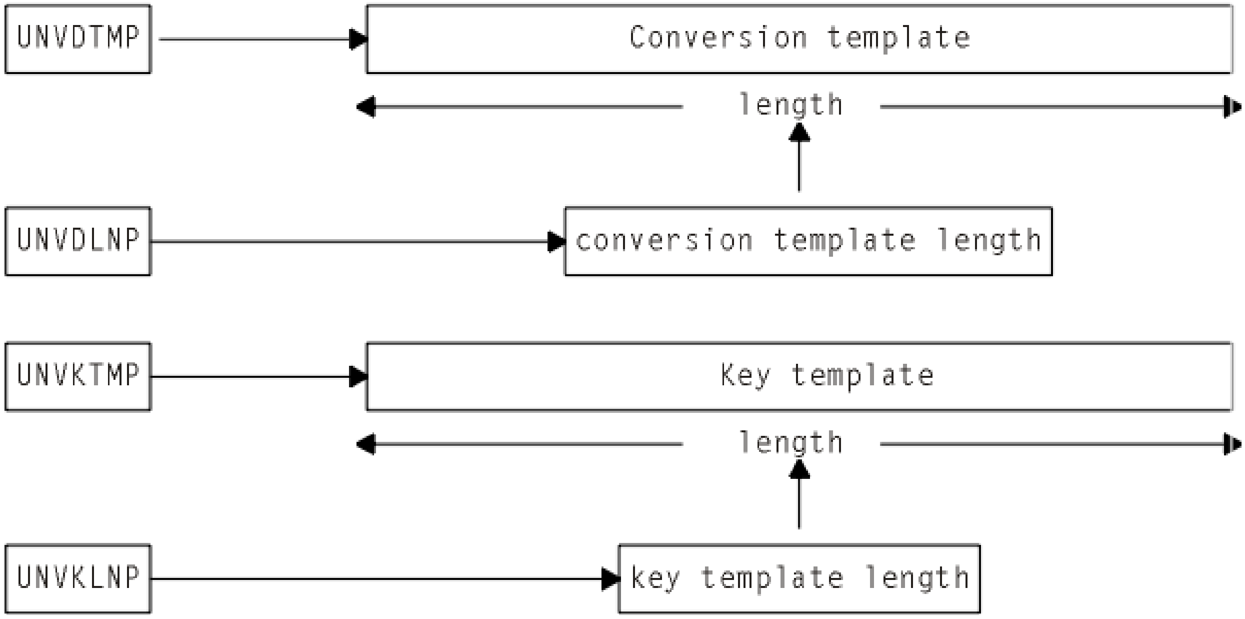


Figure 72. Parameter fields and the conversion templates

Each type of template consists of field conversion records, one for each field in the data record or key. Each field conversion record has the same layout, shown under "Field conversion records" on page 249, and mapped by a supplied DSECT, DFHCNVDS (see "DFHCNVDS, DSECT for field conversion records" on page 251). Figure 73 on page 249 shows the relationship between a template, field conversion records, and DFHCNVDS. The figure shows DFHCNVDS overlaying the first field conversion record in a template for a data record or key with six fields.

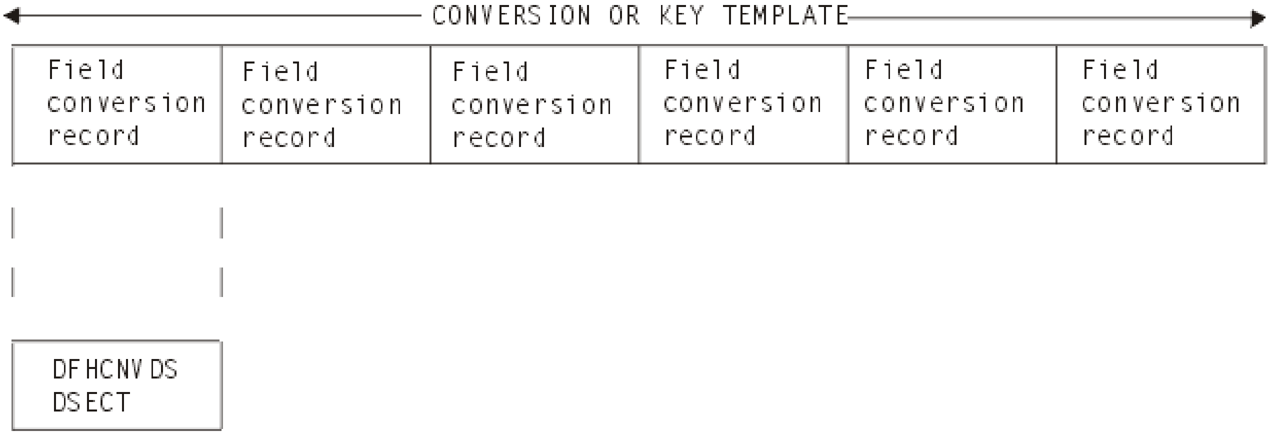


Figure 73. Field conversion records and a conversion or key template

Field conversion records

The layout of the field conversion records are described.

A field conversion record has the following layout:

Table 22. Layout of a field conversion record

CNVRLEN	CNVRTYPE	Reserved	CNVDATTY	CNVDATAO	CNVDATAL
Record length	Record type	Reserved	Data type	Data offset	Data length
Byte 1	Byte 2	Byte 3	Byte 4	Byte 5-8	Byte 9-12

In Table 22 on page 249, record length and type refer to the length and type of the field conversion record. The names in the first row are those used in the DSECT DFHCNVDS, which maps field conversion records (see “DFHCNVDS, DSECT for field conversion records” on page 251). A template has as many field conversion records as are necessary to describe all the fields in the data record or key.

For DFHUCNV, CNVRLEN is X'0C' and CNVRTYPE is always X'04' (field). DFHUCNV must interpret CNVDATTY values in the range X'50' through X'80' according to user specifications, and apply the appropriate conversions. DFHUCNV should ignore fields with CNVDATTY values outside the range X'50' to X'80'.

EQUATEs in DFHCNVDS

DFHCNVDS contains EQUATEs that are useful in your conversion program.

For resource type addressed by the parameter list

```
           CNVFC           FILE CONTROL
           CNVTS           TEMPORARY STORAGE
           CNVTD           TRANSIENT DATA
           CNVIC           INTERVAL CONTROL
           CNVPC           PROGRAM CONTROL
```

For field type in the template

Two additional EQUATEs, DTUSRMIN and DTUSRMAX, define the limits of the range of data types (X'50' to X'80') reserved for user definition. Ensure that DFHUCNV can deal with any data type in this range that can be used in your installation.

```
           DTBIN           BINARY
           DTPD            PACKED DECIMAL
           DTCHAR          CHARACTER
           DTMIX           MIXED CHARACTER
           DTDBCS          DBCS CHARACTER
           DTNUM           INTEL INTEGER
```

The supplied DFHUCNV program contains examples of the use of CNVTS, DTUSRMIN, and DTUSRMAX—see “Supplied user-replaceable conversion program” on page 251.

DFHCNVDS, DSECT for field conversion records

```
DFHCNVDS DSECT
*
*        PROVIDES A MAPPING OF THE FIELD CONVERSION RECORDS USED
*        WHEN DECIDING WHETHER TO CONVERT USER DATA.
*        A SET OF FIELD DEFINITIONS MAKE UP A TEMPLATE
*
CNVRLEN  DS    AL1                 LENGTH OF THIS RECORD
CNVRTYPE DS    XL1                 TYPE OF RECORD
*
*        EQUATES FOR RECORD TYPES
*
CNVTFLD  EQU   X'04'               FIELD (ONLY VALID TYPE IN
*                                  TEMPLATE)
CNVOVLY  DS    0H
**
**
         ORG   CNVOVLY             TYPE FIELD
         DS    XL1                 RESERVED
CNVDATTY DS    XL1                 DATA TYPE
*
*        EQUATES FOR DATA TYPES
*
DTBIN    EQU   X'01'               BINARY
DTPD     EQU   X'02'               PACKED DECIMAL
DTCHAR   EQU   X'03'               CHARACTER
DTMIX    EQU   X'04'               MIXED CHARACTER
DTDBCS   EQU   X'05'               DBCS
DTNUM    EQU   X'06'               NUMERIC
DTUSRMIN EQU   X'50'               MINIMUM USER DATA TYPE
DTUSRMAX EQU   X'80'               MAXIMUM USER DATA TYPE
*
CNVDATAO DS    AL4                 DATA OFFSET
CNVDATAL DS    AL4                 DATA LENGTH
**
*
*        EQUATES FOR RESOURCE TYPES
*
CNVFC    EQU   X'01'               FILE CONTROL
CNVTS    EQU   X'02'               TEMP STORAGE
CNVTD    EQU   X'03'               TRANS DATA
CNVIC    EQU   X'05'               INTERVAL CONTROL
CNVPC    EQU   X'06'               PROGRAM CONTROL
```

Figure 74. DFHCNVDS, DSECT that maps conversion/key templates passed to DFHUCNV

Supplied user-replaceable conversion program

The supplied version of DFHUCNV checks for a resource type of TS. If it finds one, it scans down the passed template looking for fields defined with a type in the user-data range. If any are present, DFHUCNV converts them as characters; you can rewrite the conversion code to your own requirements.

Study the supplied version of DFHUCNV and its introductory comments to enable you to write your own conversion program. Your program must be able to handle 31 bit addresses.

The supplied sample is defined to CICS with program attribute CONCURRENCY(THREADSAFE). Any code added to the sample must be threadsafe as the program might be started on an open TCB. Alternatively you can change the program definition to specify CONCURRENCY(QUASIRENT) but this change can produce a TCB switching overhead.

Chapter 4. Administering connections between CICS systems

You can manage definitions of MRO, IPIC, and APPC parallel-session connections between CICS Transaction Server for z/OS systems using connection definitions.

Important: For considerations that apply to other types of connection, see Connections that do not fully support shunting.

Recovery information for a remote system is largely independent of the connection definition for the system. This allows you to manage (for example, modify) connection definitions independently of any recovery information that may be outstanding. However, in some cases the connection definition holds important information, which means that it must be kept, unmodified, until any recovery between the systems is complete.

MRO and IPIC connections to CICS TS for z/OS systems

For connections to other CICS Transaction Server for z/OS systems, the connection definition contains no recovery information. You can modify connections without regard to recovery, provided that the netname of the connection remains the same.

If a connection definition is lost at a cold start, use the **`CEMT INQUIRE UOWLINK`** RESYNCSTATUS(UNCONNECTED) command to discover whether CICS retains any recovery information for the previously-connected system. This command will tell you whether CICS contains any tokens (UOW-links) associating UOWs with the lost connection definition. If there are UOW-links present, you can either:

- Reinstall a suitable connection definition based on the UOW-link attributes and reestablish the connection.
- If you are certain that the associated UOW information is of no use, use the **`SET UOWLINK`**(xxxxxxx) ACTION(DELETE) command to delete the UOW-link. (You may need to use the SET UOW command to force an indoubt UOW to commit or back out before the UOW-links can be deleted.)

You can use the same UOWLINK commands if a connection has been discarded.

Before discarding a connection, you should use the **`INQUIRE CONNECTION`** RECOVSTATUS command to check whether there is any recovery information outstanding. If there is recovery information outstanding, you should discard the connection only if there is no possibility of achieving a successful resynchronization with the partner. In this exceptional circumstance, you can use the **`SET CONNECTION`** UOWACTION command to force indoubt units of work before discarding the connection.

APPC parallel-session connections to CICS TS for z/OS systems

APPC parallel-session connections in a CICS Transaction Server for z/OS system that is not registered as a member of a z/OS Communications Server generic resource contain no recovery information and can be managed in the same way as MRO connections to CICS TS for z/OS systems.

APPC connections to and from z/OS Communications Server generic resources

If CICS is a member of a z/OS Communications Server generic resource group, the local z/OS Communications Server may have an affinity which directs any new binds from a partner to this same local system.

You must not end the affinity held by z/OS Communications Server if there is any possibility that resynchronization with the partner may be needed; if you do, binds (and subsequent resynchronization messages) may be directed to a different member of the generic resource. In most cases, it is safest to allow the APPC connection quiesce protocol to end the affinities automatically—see APPC connection quiesce processing.

CICS prevents the execution of the SET CONNECTION ENDAFFINITY command if a logname has been received from the partner system, because this is the condition under which the partner may begin recoverable work and start resynchronization. The discarding of a connection is also prevented, because its loss means that the logname is no longer visible. If you intend ending affinities, you should do it *before* shutting down CICS before a cold start, because a cold start restores a logname without the associated connection. Ending affinities without removing the logname can cause exchange logname failures later.

For further information about affinities and how to end them, see Ending affinities.

Managing connection definitions

For members of a generic resource, the connection definition is the only way (using the INQUIRE and SET CONNECTION RECOVSTATUS commands) of safely managing lognames and affinities.

Connections can be discarded only if their recovery status (RECOVSTATUS) is NORECOVDATA. You can use the SET CONNECTION RECOVSTATUS command to set a connection's recovery status to NORECOVDATA if neither the local system nor the partner has any indoubt units of work dependent on the other. A simple and safe test is that neither system's connection to the other should have a status of RECOVSTATUS(RECOVDATA). If this test succeeds, you can issue SET CONNECTION NORECOVDATA on both, and SET CONNECTION ENDAFFINITY on the generic resource members.

Intercommunication and z/OS Communications Server persistent sessions

The use of z/OS Communications Server persistent sessions support has some implications for intersystem communication.

For definitive information about CICS support for z/OS Communications Server persistent sessions, see Recovery with z/OS Communications Server persistent sessions .

The use of z/OS Communications Server persistent sessions has implications for DTP applications that use the APPC protocol. These implications are described in Effect of z/OS Communications Server persistent sessions support for DTP conversations on APPC sessions.

Interconnected CICS environment, recovery and restart

CICS systems can be interconnected using MRO, LU6.1, or LU6.2 connections and sessions. Recovery and restart behavior varies depending on the session type and whether or not z/OS Communications Server persistent sessions support is used.

MRO sessions

MRO connections cannot persist across CICS failures and subsequent emergency restarts.

LU6.1 sessions

If a CICS region fails in a multisystem environment, all the LU6.1 sessions that are connected to it are held in recovery pending state until it is restarted with an emergency restart or until the expiry of the persistent session delay interval. In either case, the LU6.1 sessions are then unbound. They need to be reacquired before they can be used again.

Slightly different symptoms of the CICS failure are presented to the systems programmer or operator, depending on whether persistent sessions support is used. In systems without persistent sessions support, all the LU6.1 sessions unbind immediately after the failure.

In a system with persistent session support, the LU6.1 sessions are not unbound until the emergency restart, if this occurs within the persistent session delay interval, or the expiry of the persistent session delay interval. Consequently, these sessions might take a longer time to be unbound.

LU6.2 sessions

LU6.2 sessions that connect different CICS systems are capable of persistence across the failure of one or more of the systems and a subsequent emergency restart within the persistent session delay interval.

However, these sessions are unbound in certain circumstances, even if persistent sessions are supported in your system. The following sessions are unbound after a CICS failure and emergency restart, even if you have defined them to be persistent:

- Sessions for which no catalog entry is found:
  - Autoinstalled LU6.2 parallel sessions.
  - Autoinstalled LU6.2 single sessions initiated by BIND requests.
  - Autoinstalled LU6.2 single sessions initiated by z/OS Communications Server VTAM CINIT requests, if the **AIRDELAY** system initialization parameter is set to zero. (**AIRDELAY** specifies the interval that elapses after an emergency restart before autoinstalled terminal entries that are not in session are deleted.)

    In other words, the only autoinstalled LU6.2 sessions that are not unbound are single sessions initiated by CINIT requests, and then only if **AIRDELAY** is greater than zero.
- *All* sessions on an LU6.2 connection to a failing TOR, where, on one or more of the sessions, an AOR has function-shipped an ATI request to the TOR, because the request is associated with a terminal owned by the TOR. ATI-initiated transaction routing is described in Traditional routing of transactions started by ATI.
- *All* sessions on an LU6.2 connection, where, on one or more of the sessions, transaction routing by means of CRTE is taking place but no conversation is in progress at the point of the failure. Where a conversation is in progress, a DEALLOCATE(ABEND) is sent to the partner of the failing CICS.

After the failure of CICS in an LU6.2 interconnected environment, and a subsequent emergency restart within the persistent session delay interval, transaction CLS1 (CNOS) is not run *unless* one side of the connection issued a CNOS request to zero or the connection was in the process of CNOS negotiation at the time of the failure.

The failing system runs transaction CLS2 (XLN, exchange log names) as soon as it can after emergency restart within the persistent session delay interval. CLS2 must run before any further synclevel 2 conversations can be processed by either of the connected systems.

Administering CICS in a multiregion environment

If CICS is operated in a multiregion environment, communication between regions can be supported by either multiregion operation (MRO) or intersystem communication (ISC).

Before you begin

The multisystem environment must already have been defined. You must also be familiar with the concepts related to CICS intercommunications.

About this task

You use MRO to support communication between two or more CICS regions running in the same MVS image. MRO also supports communication between separate MVS images within the same sysplex. MRO uses the CICS internal facilities and protocols and is entirely independent of SNA communications.

You use ISC to support communication between two or more systems in the same host, or to support communication in different hosts. ISC uses an SNA access technology such as ACF/SNA. The hosts can be different operating systems, and the communicating systems can be other than CICS. For example, a CICS region running in an MVS image can communicate with a CICS Transaction Server for VSE partition. Alternatively, on two MVS platforms, the communicating systems might be CICS and IMS.

For more information about operating CICS in a multiregion environment, see Multiregion operation.

Chapter 5. Developing in an intersystem environment

You can develop applications for CICS that handle function shipping, DPL, asynchronous processing, transaction routing and CICS-to-IMS intercommunication.

It contains the following chapters:

- "Application programming overview" on page 257
- "Application programming for CICS function shipping" on page 258
- " Application programming for CICS DPL " on page 260
- "Application programming for asynchronous processing" on page 264
- "Application programming for CICS transaction routing" on page 264
- "CICS-to-IMS applications" on page 267.

For guidance about application design and programming for distributed transaction processing, see Distributed transaction processing overview

Application programming overview

Application programs that are designed to run in the CICS intercommunication environment can use one or more of these facilities.

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

The application programming requirements for each of these facilities are described separately in the remaining chapter s of this part . If your application program uses more than one facility, you can use the relevant chapter as an aid to designing the corresponding part of the program. Similarly, if your program uses more than one intersystem session for distributed transaction processing, it must control each individual session according to the rules given for the appropriate session type.

Terminology

The following terms are sometimes used without further explanation in the remaining chapter s of this part :

Principal facility
This term means the terminal or session that is associated with your transaction when the transaction is initiated. CICS commands, such as SEND or RECEIVE, that do not explicitly name a facility, are taken to refer to the principal facility. Only one principal facility can be owned by a transaction.

Alternate facility
In distributed transaction processing, a transaction can acquire the use of a session to a remote system. This session is called an alternate facility. It must be named explicitly on CICS commands that refer to it. A transaction can own more than one alternate facility.

Other intersystem sessions, such as those used for function shipping, are not owned by the transaction, and are not regarded as alternate facilities of the transaction.

Front-end and back-end transactions
In distributed transaction processing, a pair of transactions converse with one another. The *front-end transaction* is initiated first, acquires a session to the remote system, and causes the *back-end transaction* to be initiated.

Note that a transaction can at the same time be the back-end transaction on one conversation and the front-end transaction on one or more other conversations.

Problem determination

Application programs that use CICS intercommunication facilities are liable to be subject to error conditions not experienced in single-CICS systems.

Where the resource is remote, the function manager is also remote, so the transaction abend is suffered by the remote transaction. This in turn causes the local transaction to be abended with a transaction abend code of AIPM (for communication through IPIC), ATNI (for communication through z/OS Communications Server), or AZI6 (for communication through MRO) rather than the particular code used in abending the remote transaction. However, the remote system sends the local CICS system an error message identifying the reason for the remote failure. This message is sent to the local CSMT destination. Therefore, if an application program uses HANDLE ABEND to continue processing when abends occur while accessing resources, it is unable to do so in the same way when those resources are remote.

Trace and memory dump facilities are defined in both local and remote CICS systems. When the remote transaction is abended, its CICS transaction dump is available at the remote site to assist in locating the reason for an abend condition.

Applications to be used with remote systems should be well tested to minimize the possibility of failing when accessing remote resources. A “remote test system” can reside in the same processor as the local system and so be tested in a single location where the transaction dumps from both systems, and the corresponding trace data, are readily available. The two transactions can be connected through MRO or through the z/OS Communications Server application-to-application facility.

Application programming for CICS function shipping

This chapter contains the following topics:

- “Introduction to programming for function shipping” on page 258
- “File control” on page 259
- “DL/I” on page 259
- “Temporary storage” on page 259
- “Transient data” on page 259
- “Function shipping exception conditions” on page 260.

Introduction to programming for function shipping

If you are writing a program to access resources in a remote system, you code it in much the same way as if the resources were on the local system. Function shipping is available by using **EXEC CICS** commands, DL/I calls or EXEC DLI commands.

The commands that you can use to access remote resources are:

- File control commands
- DL/I calls or EXEC DLI commands
- Temporary storage commands
- Transient data commands.

For information about interval control commands, see “Application programming for asynchronous processing” on page 264.

Your application can run in the CICS intercommunication environment and make use of the intercommunication facilities without being aware of the location of the resource being accessed. The location of the resource is specified in the resource definition. Optionally, you can use the SYSID option on

EXEC commands to select the system on which the command is to be executed. In this case, the resource definitions on the local system are not referenced, unless the SYSID option names the local system.

When your application issues a command against a remote resource, CICS ships the request to the remote system, where a mirror transaction is initiated. The mirror transaction executes the request on your behalf, and returns any output to your application program. The mirror transaction is like a remote extension of your application program. For more information about this mechanism, read CICS function shipping.

Although the same commands are used to access both local and remote resources, there are restrictions that apply when the resource is remote. Also, some errors that do not occur in single systems can arise when function shipping is being used. For these reasons, you should always know whether resources that your program accesses can possibly be remote.

File control

Function shipping allows you to access files located on a remote system.

If you use the SYSID option to access a remote system directly, you must observe the following two rules:

1. For a file referencing a keyed data set, KEYLENGTH must be specified if RIDFLD is specified, unless you are using relative byte addresses (RBA) or relative record numbers (RRN).

   For a remote BDAM file, where the DEBKEY or DEBREC options have been specified, KEYLENGTH must be the total length of the key.
2. If the file has fixed-length records, you must specify the record length (LENGTH).

These rules also apply if the definition of the file to this CICS does not specify the appropriate values.

DL/I

You can use function shipping to access an IMS Database Manager subsystem that is associated with a remote CICS system, or a database associated with a remote CICS Transaction Server for VSE system.

Temporary storage

You can use function shipping to send data to or receive data from temporary storage queues located on remote systems.

The systems programmer can use TSMODEL resource definitions to define temporary storage models that direct matching EXEC CICS requests to remote systems. TSMODEL resource definitions do not support the use of the SYSID option on the WRITEQ TS, READQ TS, and DELETEQ TS commands to specify the remote system explicitly.

For MRO and IPIC sessions, the MAIN and AUXILIARY options of the WRITEQ TS command can be used to select the required type of storage.

For APPC sessions, the MAIN and AUXILIARY options are ignored; unless a TSMODEL or exit directs it otherwise, auxiliary storage is always used in the remote system.

Transient data

Function shipping allows you to access intrapartition or extrapartition transient data queues located on remote systems. Definitions of remote transient data queues can be made by the system programmer. You can, however, use the SYSID option on the WRITEQ TD, READQ TD, and DELETEQ TD commands to specify the system on which the request is to be executed.

If the remote transient data queue has fixed-length records, you must supply the record length if it is not specified in the transient data resource definition that has been installed.

Function shipping exception conditions

Requests that are shipped to a remote system can raise any of the exception conditions for the command that can occur if the resource is local.

Some additional conditions apply only when the resource is remote.

Remote system not available

The SYSIDERR condition is raised in the application program under certain situations.

These are the following situations:

- The link to the remote system is out of service.
- The named system is not defined. This error should not occur in a production system unless the application is designed to obtain the name of the remote system from a terminal operator.
- The link to the remote system is busy, and the maximum number of queued requests specified on the QUEUELIMIT option of the CONNECTION or IPCONN resource definition has been reached.
- The link to the remote system is busy, the maximum number of queued requests has not been reached, but your XZIQUE, XISCONA or XISQUE global user exit program specifies that the request should not be queued. For programming information about the XZIQUE and XISCONA exits, see Intersystem communication program exits, XISCONA, XISLCLQ, and XISQLCL . The XISQUE global user exit program is used for IPIC connections, for further information about XISQUE see XISQUE exit for managing IPIC intersystem queues.

The default action for the SYSIDERR condition is to terminate the task abnormally.

Invalid request

The ISCINVREQ condition occurs when the remote system indicates a failure that does not correspond to a known condition. The default action is to terminate the task abnormally.

Mirror transaction abend: remote resource

An application request against a remote resource can cause an abend in the mirror transaction in the remote CICS. For example, a deadlock timeout causes an abend of the mirror transaction with an abend code of ATSC.

In these situations, the application program also abends, but with one of the following abend codes:

- AIPM for IPIC connections
- ATNI for ISC connections
- AZI6 for MRO connections

CICS logs the error condition in an error message that is sent to the CSMT destination. Any HANDLE ABEND command that the application issues cannot identify the original cause of the condition and take explicit corrective action. Corrective action might have been possible if the resource had been local. An exception occurs in MRO function shipping if the mirror transaction abends with a DL/I program isolation deadlock; in this case, the application abends with the normal deadlock abend code (ADCD).

The ATNI abend caused by a mirror transaction abend is not related to a terminal control command, and therefore the TERMERR condition is not raised.

For information about the AEZA or AEZC abend, see “Mirror transaction abend: DPL” on page 263.

Application programming for CICS DPL

This chapter contains the following topics:

- “Introduction to DPL programming” on page 261
- “The client program” on page 261

- “The server program” on page 261
- “DPL exception conditions” on page 262.

Introduction to DPL programming

You can use CICS distributed program link (DPL) to link to server programs located on a remote system.

A client program that runs in a CICS Transaction Server for z/OS region can link to one or more server programs that run in remote CICS regions. The remote regions can be CICS Transaction Server for z/OS systems, but might not be. See Introduction to CICS intercommunication for a list of systems with which CICS Transaction Server for z/OS can communicate.

You can write DPL programs in PL/I, C, COBOL, or assembler language.

There are two sides (programs) involved in DPL: the client program and the server program. For further description, see Overview of DPL . The following information describes the actions that each program must take to implement DPL.

The client program

If you are writing a client program to link to a server program in a remote system, you code it in much the same way as if the server program were on the local system.

Your client program can run in the CICS intercommunication environment and make use of intercommunication facilities without being aware of the location of the server program being linked to. The location of the server program is specified by the program resource definition or the dynamic routing program. Optionally, you can use the SYSID option on the LINK command to select the system on which the command is to be executed.

When your client program issues a LINK command against a server program, CICS ships the request to the remote system, where a mirror transaction is initiated. The mirror transaction executes the LINK request on your behalf, thereby causing the server program to run. When the server program issues a RETURN command, the mirror transaction returns any communication area data to your client program. The mirror transaction is like a remote extension of your application program. For more information about this mechanism, read CICS distributed program link.

Although the same command is used to access both local and remote server programs, there are restrictions that apply when the server program is remote. Also, some errors that do not occur in single systems can arise when DPL is being used. For these reasons, you should always find out whether the server program to which your client program links is remote. If there is any possibility of the server program being remote, the client program should include the additional checks for the exception conditions that can be returned by a remote server program.

Failure of the server program

If the server program fails, the ABEND condition and an abend code are returned to the client program. The client program therefore also terminates abnormally, unless it has issued the HANDLE ABEND command before issuing the LINK command.

The server program

Permitted commands

The **`EXEC CICS`** commands that a DPL server program can issue are limited to a subset of the CICS API. The DPL subset also includes **`EXEC DLI TERM`** , and the **`CALL`** interface equivalent.

For details of the restricted DPL subset, see Exception conditions for LINK command.

Syncpoints

If the server program was started by a LINK command that specified the SYNCONRETURN option, it is able to issue a syncpoint.

If it does, this does **not** commit changes made by the client program. For changes to be committed across the distributed unit of work, the client program must issue the syncpoint. The client program can also backout changes across the distributed unit of work, provided that the server program has not already committed its changes.

The server program can find out how it was started, and therefore whether it is allowed to issue independent syncpoint requests, by issuing the ASSIGN STARTCODE command. This command returns the following values relevant to a DPL server program:

- 'D' if the program was started by a LINK request **without** the SYNCONRETURN option, and cannot therefore issue SYNCPOINT requests.
- 'DS' if the program was started by a LINK request **with** the SYNCONRETURN option, and can therefore issue SYNCPOINT requests. However, the server program need not issue a syncpoint request explicitly, because CICS takes a syncpoint as soon as the server program issues the RETURN command.
- Values other than 'D' and 'DS' if the program was not started by a remote LINK request.

DPL exception conditions

LINK requests that are shipped to a remote system can raise any of the exception conditions for the command that can occur if the server program is local.

Some additional conditions apply only when the server program is remote.

Remote system not available

When the remote system is unavailable, the SYSIDERR condition can be raised in the client program.

The reasons for raising the SYSIDERR condition are the same as those described for function shipping. See “Remote system not available” on page 260.

The default action for the SYSIDERR condition is to terminate the task abnormally.

Server's work backed out

If the client program issues the LINK command with the SYNCONRETURN option, the mirror program issues a syncpoint as soon as the server program terminates successfully.

It is possible for this syncpoint to fail. If this happens, the ROLLEDBACK condition is returned to the client program. The work done by the server program will also be backed out, *unless the server program has already committed the work by issuing its own syncpoint request*.

Multiple links to the same server region

When a client program issues a LINK command with the SYNCONRETURN option, the mirror transaction terminates as soon as control is returned to the client program. It is therefore possible for the client program to issue a subsequent LINK command to the same server region.

However, when a client program issues a LINK command without the SYNCONRETURN option, the mirror transaction is suspended pending a sync point request from the client region. The client program can issue subsequent LINK commands to the same server region, as long as the SYNCONRETURN option is omitted and the TRANSID value is not changed. A subsequent LINK command with the SYNCONRETURN option, or with a different TRANSID value, is unsuccessful unless it is preceded by a SYNCPOINT command.

Note: Similar considerations apply if the client program sends function shipping requests to the server region, and the mirror for the function shipping request is suspended. For example:

```
EXEC CICS LINK PROGRAM('PGA') SYSID(SERV)
EXEC CICS SYNCPOINT
EXEC CICS READQ TS QUEUE('RQUEUE') SYSID(SERV)
EXEC CICS LINK PROGRAM('PGB') SYSID(SERV) TRANSID(TRN1)
```

The last LINK command fails if, for example, MROLRM=YES is specified in the CICS server region (SERV). This is because the mirror used for the READQ TS command is still around. For this example sequence of commands to work, the client program must issue a SYNCPOINT after the READQ TS command. Alternatively, you could set the MROLRM system initialization parameter to NO in the server region. For detailed information about using DPL and function shipping requests in the same program, see Programming considerations for distributed program link.

These errors are indicated by the INVREQ and PGMIDERR conditions.

On the INVREQ condition, an accompanying RESP2 value of 14 indicates that a sync point is necessary before the failed LINK command can be successfully attempted. A RESP2 value of 15 indicates that the TRANSID value is different from that of the linked mirror transaction. A RESP2 value of 16 indicates that a TRANSID value of spaces (blanks) was specified on the LINK command. A RESP2 value of 17 indicates that a TRANSID value of spaces (blanks) was supplied by the dynamic routing program.

On the PGMIDERR condition, an accompanying RESP2 value of 25 indicates that the dynamic routing program rejected the link request.

Mirror transaction abend: DPL

If the mirror program (as opposed to the server program) abends, or the session with the server region fails, the TERMERR condition is returned to the client program.

The CICS-supplied mirror transactions use 31-bit storage (above 16 MB but below 2 GB). If an **`EXEC CICS LINK`** command is issued over DPL for an AMODE(24) application, an AEZA or AEZC abend will occur. To avoid this situation, do one of the following:

- Define your own mirror transaction that uses 24-bit storage. For example, you can copy a CICS-supplied mirror transaction, then specify the TASKDATALOC(BELOW) attribute. You can use the **`CEDA COPY`** command or the **`DFHCSDUP COPY`** command to make a copy of a CICS-supplied mirror transaction in another group.
- Modify the application so that it is AMODE(31) and update the appropriate program definition.

Multiple updates to a recoverable resource by the same distributed UOW

In a non-DPL environment, it is possible for multiple programs within one unit of work (UOW) to update the same recoverable resource.

For example, `program1` might update `Record1` in a recoverable file, then link to `program2` , which could update the same record, `Record1` , in the same file. This might not be good programming practice but it does work, because CICS considers the owner of the resource to be the task, not the program.

However, in a DPL environment, where the programs involved are running in different CICS regions, it is not possible for multiple programs to update the same recoverable resource within the same UOW. Using the same example, `program1` updates `Record1` in a recoverable file, then links to `program2` , which runs under a mirror task in another region. If `program2` function-ships a file control request to update `Record1` in the same file, the request hangs. It hangs because the mirror task that processes the file control request of `program2` cannot get the record lock for `Record1` . The lock is owned by the task under which `program1` is running. Even though the file control mirror task and the task under which `program1` is running are part of the same distributed UOW, CICS does not allow the update. This is because CICS uses the task, not the distributed UOW, as the basis for locking recoverable resources.

Application programming for asynchronous processing

This section discusses the application programming requirements for CICS -to-CICS asynchronous processing.

The general information given for CICS transactions that use the **START** or **RETRIEVE** commands is also applicable to CICS-to- IMS communication.

A description of the concepts of asynchronous processing is given in Asynchronous processing . It is assumed that you are familiar with the concepts of CICS interval control. For programming information about the use of **EXEC CICS** commands for interval control, see START.

Starting a transaction on a remote system

You can start a transaction on a remote system by issuing an EXEC CICS START command just as though the transaction were a local one.

About this task

Generally, the transaction has been defined as remote by the system programmer. You can, however, name a remote system explicitly in the SYSID option. This use of the START command is thus essentially a special case of CICS function shipping.

If your application requires you to specify the time at which the remote transaction is to be initiated, remember that the remote system may be in a different time zone. The use of the INTERVAL form of control is preferable under these circumstances.

Exception conditions for the START command

The exception conditions that can occur as a result of issuing a START request for a remote transaction depend on whether the NOCHECK performance option is specified on the START command.

If NOCHECK is not specified, exception conditions are raised according to the rules for function shipping (see "Function shipping exception conditions" on page 260).

If NOCHECK is specified, no conditions are raised as a result of the remote execution of the START command. However, the SYSIDERR condition occurs if no link to the remote system is available, unless the system programmer arranged for local queuing of start requests (see Local queuing of START commands).

Retrieving data associated with a remotely-issued start request

The RETRIEVE command is used to retrieve data that has been stored for a task as a result of a remotely-issued start request. This is the only available method for accessing such data.

About this task

As far as your transaction is concerned, there is no distinction between data stored by a remote start request and data stored by a local start request, and the normal considerations for use of the RETRIEVE command apply.

Application programming for CICS transaction routing

In general, if you are writing a transaction that may be used in a transaction routing environment, you can design and code it just as you would for a single CICS system.

There are, however, a number of restrictions that you must be aware of, and these are described in this chapter . The same considerations apply if you are migrating an existing transaction to the transaction routing environment.

Application programming restrictions

There are a number of restrictions and considerations when you write application programs for transaction routing.

The program can be written in PL/I, COBOL, C, or assembler language. This choice might, of course, be restricted by the terminal or session type: basic APPC conversations, for example, must be written in C or assembler language.

Basic mapping support

Any BMS maps or partition sets that your program uses must reside in the same CICS system as the program.

In a BMS routing application, a route request that specifies an operator or an operator class directs output only to the operators signed on at terminals that are owned by the system in which the transaction is executing.

The mapset name specified in the most recent SEND MAP command is saved in the TCTTE. For a routed transaction, this means that the mapset name is saved in the surrogate TCTTE and, when the routed transaction terminates, the most recently used mapset name is passed in a DETACH sequence from the AOR to the TOR.

Similarly, when a routed transaction is initiated, the most recently used mapset name is passed in an ATTACH sequence from the TOR to the AOR.

The *map* name is supported in the same way as the *mapset* name. However, some old CICS products (no longer supported) have no knowledge of map names being passed in ATTACH and DETACH sequences. When sending an ATTACH sequence, CICS Transaction Server for z/OS systems set the map name to null values in the “real” TCTTE, in case the AOR is unable to return a map name in the DETACH sequence. In other words, the TCTTE in the TOR contains a null value for the saved map name, rather than a potentially incorrect name.

The names of mapsets and maps saved in the TCTTE can be both queried and updated by the MAPNAME and MAPSETNAME options of the INQUIRE TERMINAL and SET TERMINAL commands.

Pseudoconversational transactions

A routed transaction requires the use of an interregion or intersystem (APPC) session for as long as it is running. For this reason, long-running conversational transactions are best duplicated in the two systems, or alternatively designed as pseudoconversational transactions.

Take care in the naming and definition of the individual transactions that make up a pseudoconversational transaction, because a TRANSID specified in a CICS RETURN command is returned to the terminal-owning region, where it may be a local transaction.

There is, however, no reason why a pseudoconversational transaction cannot be made up of both local and remote transactions.

The terminal

The “terminal” with which your transaction runs is represented by a terminal control table terminal entry (TCTTE).

This TCTTE, called a *surrogate TCTTE* , is in many respects a copy of the “real” terminal's TCTTE in the terminal-owning region. CICS releases the surrogate TCTTE when the transaction terminates. Subsequent tasks run using new copies of the real terminal's TCTTE.

If your program needs to discover terminal-related information, consider the following points :

- Your program should not test fields in the TCTTE directly: it should test instead the equivalent fields in the EXEC interface block (EIB).

- If the new task is started by ATI, the contents of certain terminal-related fields in the EIB are unpredictable. EIBAID, which contains the attention identifier, is always set to zeros at the start of a session.

Reviewing values returned by the `EXEC CICS ASSIGN` command in the application-owning region

Review the values returned by the PRINSYSID and USERID options when you use the **`EXEC CICS ASSIGN`** command, because the values are taken from a number of sources.

PRINSYSID

This option returns the system identifier (SYSID) of the principal facility to the transaction. The value returned is the name of the remote connection or terminal defined in this system. If the connection or terminal has been shipped, the name is the original name defined in the terminal-owning region (TOR). If the principal facility is not an APPC session, the INVREQ condition is issued.

USERID

For a routed transaction, CICS takes the user ID from one of several sources, depending on how you specified your security requirements. For more information, see Transaction routing security with LU6.2.

Table 23 on page 266 explains the value that is returned by the USERID option. The values are as follows:

- If the connection is defined with the ATTACHSEC(LOCAL) option, and SEC=YES is specified in the system initialization parameters of the application-owning region (AOR), CICS returns a value that depends on the connection type:
  - For ISC over SNA and IPIC connections, the value returned is either the USERID attribute, if this attribute is specified in the SESSIONS definition, or the value of the SECURITYNAME attribute that is specified in the CONNECTION definition.
  - For MRO connections, the RACF user ID of the TOR.
- If the connection is defined with the ATTACHSEC(LOCAL) option, and SEC=NO is specified in the system initialization parameters of the AOR, CICS returns the DFLTUSER value from the AOR.
- If the connection is defined with the ATTACHSEC(IDENTIFY) option or, for APPC connections, the VERIFY, PERSISTENT, or MIXIDPE option, and SEC=YES is specified in the system initialization parameters of the TOR, CICS returns the user ID sent at attach time.
- If the connection is defined with the ATTACHSEC(IDENTIFY) option, or, for APPC connections, the VERIFY, PERSISTENT, or MIXIDPE option, and SEC=NO is specified in the system initialization parameters of the TOR, CICS returns the DFLTUSER value from the TOR.

*Table 23. Values returned by the USERID option of **`EXEC CICS ASSIGN`**, for routed transactions*

System initialization parameter SEC value in TOR	ATTACHSEC value in CONNECTION definition		
	IDENTIFY VERIFY PERSISTENT MIXIDPE	LOCAL	
		System initialization parameter SEC=YES in AOR	System initialization parameter SEC=NO in AOR
YES	User ID sent at attach	**ISC over SNA and IPIC:** 1. USERID of session 2. SECURITYNAME of connection **MRO:** RACF user ID of TOR	DFLTUSER of AOR
NO	User ID sent at attach (DFLTUSER of TOR)		

CICS-to-IMS applications

This chapter tells you how to code CICS transactions that communicate with an IMS system.

For full details of IMS ISC, refer to the appropriate IMS publications. This chapter is intended to provide sufficient information about IMS to enable you to work with your IMS counterpart to implement a CICS-to-IMS ISC application.

The chapter contains the following topics:

- "Designing CICS-to-IMS ISC applications" on page 267
- "CICS-to-IMS applications: asynchronous processing" on page 269
- "CICS-to-IMS applications: DTP" on page 273.

Designing CICS-to-IMS ISC applications

There are many differences between CICS and IMS , both in their architecture and in their application and system programming requirements.

The design of CICS-to-IMS ISC applications involves principally CICS application programming and IMS system definition. This difference reflects where the control lies in each of the two systems.

CICS is a **direct control** system. Data entered at a terminal causes CICS to invoke the appropriate application program to process the incoming data. The data is stored, rather than queued, and the application "owns" the terminal until it completes its processing and terminates. In CICS ISC, the application program is involved with data flow protocols, with syncpointing, and, in general, with most system services.

In contrast, IMS is a **queued** system. All input and output messages are queued by the IMS control region on behalf of the related application programs and terminals. The queuing of messages and the processing of messages are therefore performed asynchronously. This is illustrated in Figure 75 on page 268.

As a result of this type of system design, IMS application programs do not have direct control over IMS system resources, nor do they become directly involved in the control of intersystem communication. IMS message switching is handled entirely in the IMS control region; the message processing region is not involved.

Data formats

Messages transmitted between CICS and IMS can have either of the following data formats.

- Variable-length variable-blocked (VLVB)
- Chain of RUs.

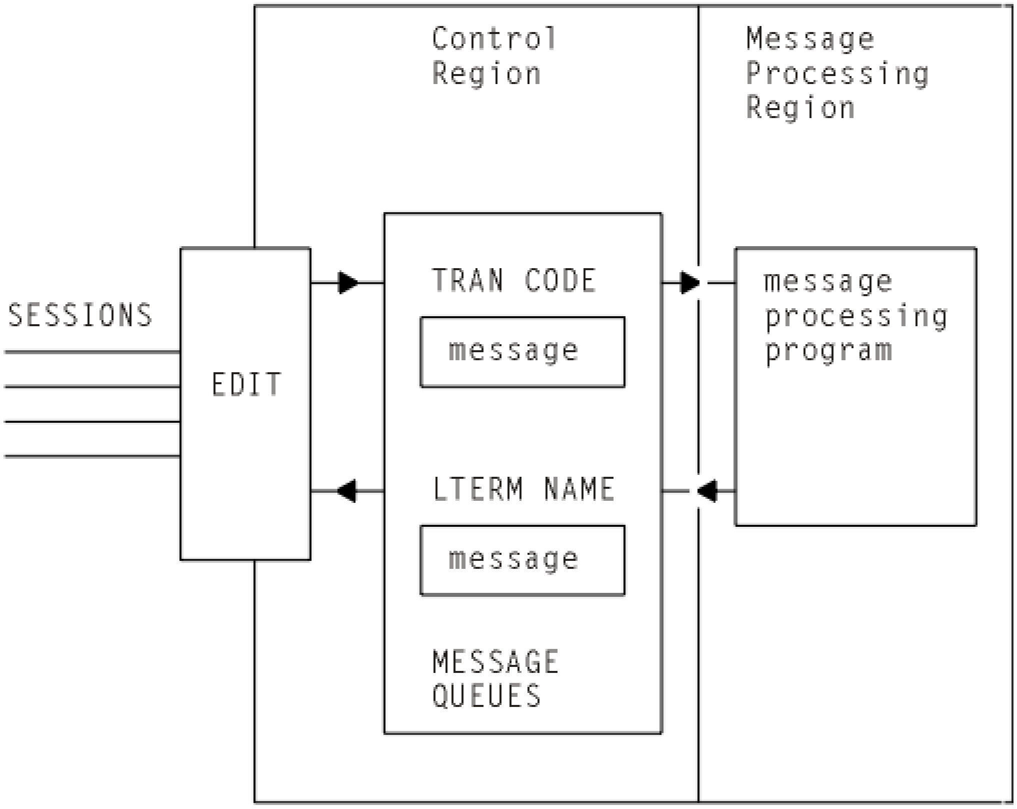


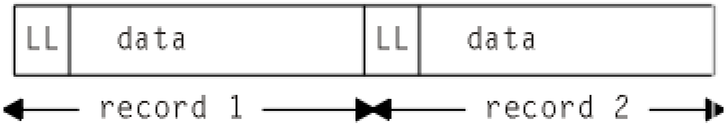
Figure 75. Basic IMS message queuing

In normal CICS communication with logical units, chain of RUs is the default data format. In IMS, VLVB is the default. In CICS-to-IMS communication, the format that is being used is specified in the LUTYPE6.1 attach headers that are sent with the initial data.

Variable-length variable-blocked

In VLVB format, a message can contain multiple records.

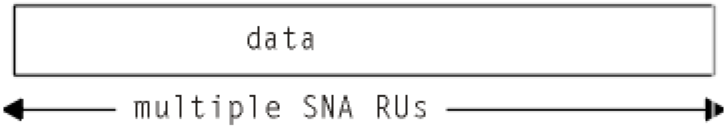
Each record is prefixed by a two-byte length field, as shown here.



In CICS, the I/O area contains a complete message, which can contain one or more records. The blocking of records for output, and the deblocking on input, must be done by your CICS application program.

Chain of RUs

In this format, which is the most common CICS format, a message is transmitted as multiple SNA RUs, as shown here.



In CICS, the I/O area contains a complete message.

Forms of intersystem communication with IMS

In an application that involves communication between CICS and IMS, the intersystem communication must be initiated by one or other of the two systems. For example, if a CICS terminal operator initiates a CICS transaction that is designed to obtain data from a remote IMS system, the intersystem communication for the purposes of this application is initiated by CICS.

There are three forms of CICS to IMS communication that must be considered:

1. Asynchronous processing using CICS START and RETRIEVE commands
2. Asynchronous processing using CICS SEND LAST and RECEIVE commands
3. Distributed transaction processing (that is, synchronous processing) using CICS SEND and RECEIVE commands.

The basic differences between these forms of communication are described in Asynchronous processing and Distributed transaction processing overview.

The system that initiates intersystem communication for a specific application is the front-end system as far as that application is concerned. The other system is called the back-end system.

When CICS is the front-end system, it supports all three types of intersystem communication in the previous list. The form of communication that can be used for any specific application depends on the IMS transaction type, or on the IMS facility that is being initiated. For information about the forms of communication that IMS supports when it is the back-end system, see Communications and connections in IMS product documentation.

When IMS is the front-end system, it always uses asynchronous processing (corresponding to the CICS START and RETRIEVE interface) to initiate communication with CICS.

CICS-to-IMS applications: asynchronous processing

In asynchronous processing, the intersystem session is used only to pass an initiation request, together with various items of data, from one system to the other. All other processing is independent of the session that is used to pass the request.

The two application programming interfaces available in CICS for asynchronous processing are:

1. The START and RETRIEVE interface
2. The SEND and RECEIVE interface.

The START and RETRIEVE interface

The applicable forms of these commands, together with the specific meanings of the command options in a CICS-to- IMS intersystem communication environment, are given in this section.

For programming information about the CICS **START** and **RETRIEVE** “interval control” commands, see START and RETRIEVE.

CICS front end

When CICS is the front-end system, you can use CICS START and RETRIEVE commands to process IMS nonresponse mode and nonconversational transactions, message switches, and the IMS /DIS, /RDIS, and /FOR operator commands.

Note: When you issue the IMS /DIS, /RDIS, and /FOR operator commands, unless you send change direction (CD), IMS expects you to request definite response. You must do this by coding the PROTECT option on the START command.

The general command sequence for your application program is shown in Figure 76 on page 270.

After transaction TRANA has obtained an input message from the terminal, it issues a START NOCHECK command to initiate the remote IMS transaction. The START command specifies the name of the IMS editor that is to be initiated to process the message, and the IMS transaction or logical terminal (LTERM) that is to receive the message. It also specifies the name of the CICS transaction that is to receive the reply, and the name of the associated CICS terminal.

The PROTECT option must be specified on the START command to ensure delivery of the message to IMS.

The start request is not shipped until your application program either issues a SYNCPOINT command, or terminates. However, the request does not carry the syncpoint-indicator unless PROTECT was specified on the START command.

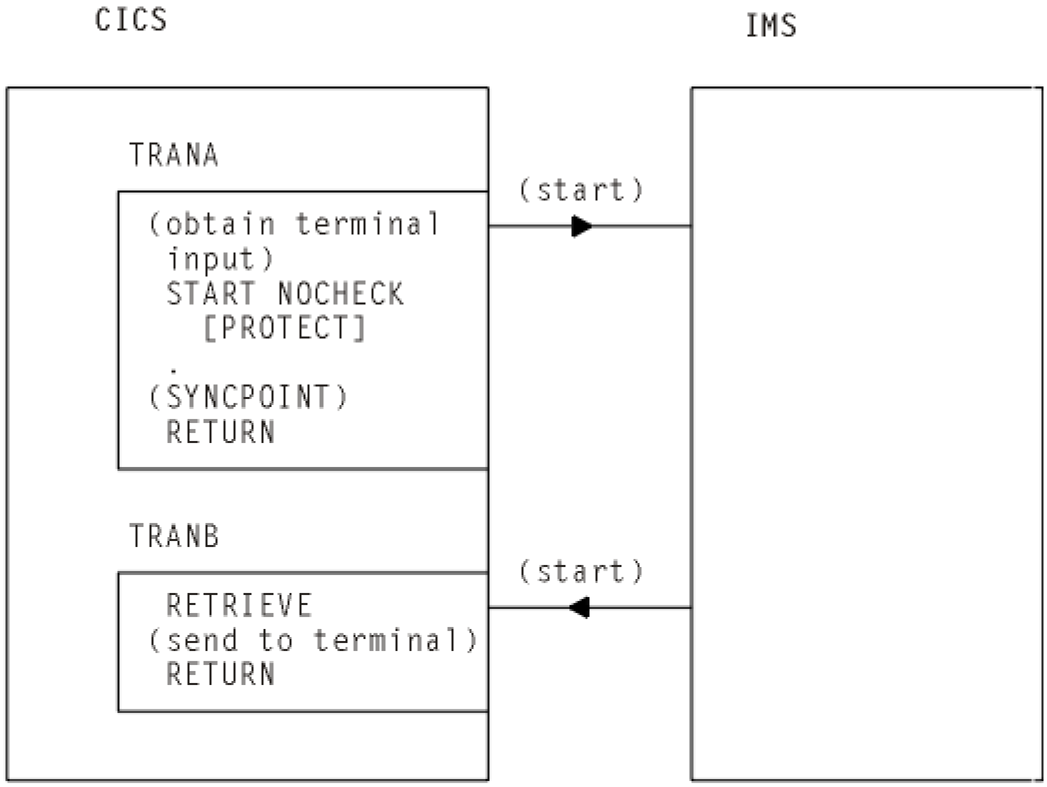


Figure 76. START and RETRIEVE asynchronous processing–CICS front end

Although CICS allows an application program to issue multiple START NOCHECK commands without intervening syncpoints (see Deferred transmission of START requests with NOCHECK option for ISC links), this technique is not recommended for CICS-to-IMS communication.

IMS sends the reply by issuing a start request that is handled in the normal way by the CICS mirror transaction. The request specifies the CICS transaction and terminal that you named in the original START command. The transaction that is started (TRANB) can then retrieve the reply by issuing a RETRIEVE command.

In the example, it is assumed that there are two separate CICS transactions; one to issue the START command and one to receive the reply and return it to the terminal. These two transactions can be combined, and there are two ways to do this:

- Write a transaction that contains both the START and the RETRIEVE processing, but that performs only one of these functions for a specific execution. The CICS ASSIGN STARTCODE command can be used to determine whether the transaction was initiated from the terminal, in which case the START processing is required, or by a start request, in which case the RETRIEVE processing is required.
- Write a transaction that, having issued the START command, issues a SYNCPOINT command to clear the start request, and then waits for the reply by issuing a RETRIEVE command with the WAIT option. The terminal is held by the transaction during this time, and CICS returns control to the transaction when input directed to the same transaction and terminal is received.

In all cases, make no assumptions about the timing of the reply, or its relationship to a specific previous request. A RETRIEVE command retrieves any outstanding data intended for the same transaction and terminal. Correlation of requests and replies is the responsibility of your application program.

IMS front end

When IMS is the front-end system, the only supported flow is the asynchronous start request. Your application program must use the RETRIEVE command to obtain the request from IMS, followed by a START command to send the reply if one is required.

The general command sequence for your application program is shown in Figure 77 on page 271.

If a reply to the retrieved data is required, your start command must specify the IMS editor and transaction or LTERM name obtained by the RETRIEVE command.

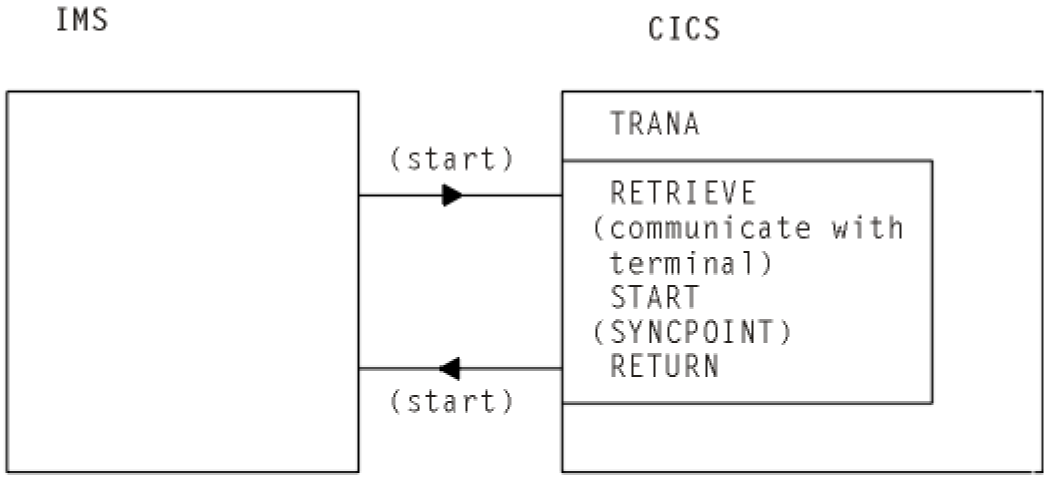


Figure 77. RETRIEVE and START asynchronous processing – IMS front end

The START command

This section shows the format of the START command that is used to schedule remote IMS transactions. Note that no interval control is possible (although it is not an error to specify INTERVAL(0)) and that the NOCHECK and PROTECT options must be specified.

```
 EXEC CICS START TRANSID(name)
[SYSID(name)]
[FROM(data-area) LENGTH(value)]
[TERMID(name)]
[RTRANSID(name)]
[RTERMID(name)]
NOCHECK
PROTECT
[FMH]
```

TRANSID(name)
Specifies the name of the IMS editor that is to be initiated to process the message. It must be an alias (not exceeding four characters) of ISCEDT, or an MFS MID name.

Alternatively, it can name the installed definition of a “remote” transaction. In this case, the SYSID option is not used. The definition of the remote transaction must name the required IMS editor in the RMTNAME option, which can be up to eight characters long.

SYSID(name)
Specifies the name of the remote IMS system. This is the name of the CONNECTION resource that defines the link to the remote system. You need this option only if you are required to name the remote system explicitly.

FROM(data-area)
Specifies the data that is to be sent. The format of the data (VLVB or chain of RUs) must match the format specified in the RECORDFORMAT attribute of the CONNECTION resource that defines the remote IMS system (see Defining connections to remote systems).

LENGTH(value)
Specifies, as a halfword binary value, the length of the data specified in the FROM option.

TERMID(name)

Specifies the primary resource name that is to be assigned to the remote process. For IMS, it is a transaction code or an LTERM name.

If this option is omitted, you must specify the transaction code or the LTERM name in the first eight characters of the data named in the FROM option. You must use this method if the name exceeds four characters (the CICS limit for the TERMID option) or if IMS password processing is required.

RTRANSID(name)

Specifies the name of the transaction that is to be invoked when IMS returns a reply to CICS. The name must not exceed four characters in length.

RTERMID(name)

Specifies the name of the terminal that is to be attached to the transaction specified in the RTRANSID option when it is invoked. The name must not exceed four characters in length.

NOCHECK

This option is mandatory.

PROTECT

Specifies that the remote IMS transaction must not be scheduled until the local CICS transaction has taken a syncpoint. PROTECT is mandatory.

FMH

Specifies that the user data to be passed to the started task contains function management headers. This option is not normally used.

The RETRIEVE command

This section shows the format of the RETRIEVE command that is used to retrieve data sent by IMS.

```
 EXEC CICS RETRIEVE
[{INTO(data-area)|SET(pointer-ref)}
LENGTH(data-area)]
[RTRANSID(data-area)]
[RTERMID(data-area)]
[WAIT]
```

INTO(data-area)

Specifies the user data area into which the data retrieved from IMS is to be written.

SET(pointer-ref)

Specifies the pointer reference to be set to the address of the data retrieved from IMS.

LENGTH(data-area)

Specifies the halfword binary length of the retrieved data.

For a RETRIEVE command with the INTO option, this must be a data area that specifies the maximum length of data that the program is prepared to handle. If the value specified is less than zero, zero is assumed. If the length of the data exceeds the value specified, the data is truncated to that value and the LENGERR condition occurs. On completion of the retrieval operation, the data area is set to the original length of the data.

For a RETRIEVE command with the SET option, this must be a data area. On completion of the retrieval operation, the data area is set to the length of the data.

RTRANSID(data-area)

Specifies an area to receive the return destination process name sent by IMS. It is either an MFS MID name chained from an output MOD, or is blank.

Your application can use this name in the TRANSID option of a subsequent START command.

RTERMID(data-area)

Specifies an area to receive the return primary resource name sent by IMS. It is either a transaction name or an LTERM name.

Your application can use this name in the TERMID option of the START command used to send the reply.

WAIT

Specifies that control is not to be returned to your application program until data is sent by IMS.

If WAIT is not specified, the ENDDATA condition is raised if no data is available. If WAIT is specified, the ENDDATA condition is raised only if CICS is shut down before any data becomes available.

The use of the WAIT option is not generally recommended, because it can cause intervening messages (not the expected reply) to be retrieved.

The asynchronous SEND and RECEIVE interface

This form of asynchronous processing is, in CICS, a special case of distributed transaction processing.

A CICS transaction acquires the use of a session to a remote system, and uses the session for a single transmission (using a SEND command with the LAST option) to initiate a remote transaction and send data to it. The reply from the remote system causes a CICS transaction to be initiated just as if it were a back-end transaction in normal DTP. This transaction, however, can issue only a single RECEIVE command, and must then free the session.

Except for these additional restrictions, you can design your application according to the rules given for distributed transaction processing later in this chapter.

The general command sequence for asynchronous SEND and RECEIVE application programs is shown in Figure 78 on page 273.

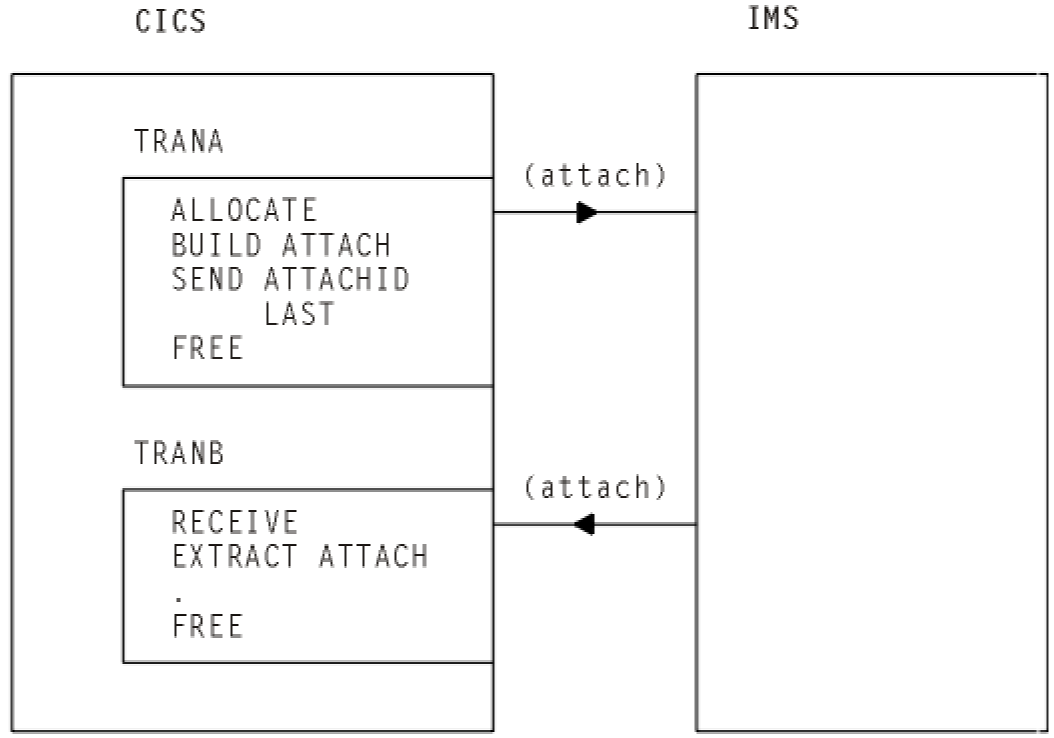


Figure 78. SEND and RECEIVE asynchronous processing – CICS front end

CICS-to-IMS applications: DTP

This section describes application programming for CICS -to- IMS distributed transaction processing (DTP).

CICS commands for CICS-to-IMS sessions

These are the commands that can be used to acquire and use CICS-to-IMS sessions.

- **ALLOCATE –** used to acquire a session to the remote IMS system.

- **BUILD ATTACH –** used to build an LUTYPE6.1 attach header that is used to initiate a transaction on a remote IMS system.
- **EXTRACT ATTACH –** used by a CICS transaction to recover information from the LUTYPE6.1 attach header that caused it to be initiated. This command is required only for SEND and RECEIVE asynchronous processing.
- **SEND, RECEIVE** , and **CONVERSE –** used by the CICS transaction to send or receive data on the session. The first SEND or CONVERSE command issued by a front-end CICS transaction must name the attach header that has been defined by the BUILD ATTACH command.
- **WAIT TERMINAL SESSION(name) –** used to ensure that CICS has transmitted any accumulated data or data flow control indicators before it continues with further processing.
- **ISSUE SIGNAL SESSION(name) –** used by a transaction that is in receive state to request an invitation to send (change-direction) from IMS.
- **FREE –** used by a CICS transaction to relinquish its use of the session.

Considerations for the front-end transaction

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

The front-end transaction is responsible for acquiring a session to the remote IMS system and initiating the remote transaction. Thereafter, the two transactions become equals. However, the front-end transaction is usually designed as the client, or driving, transaction.

Session allocation

You acquire an LUTYPE6.1 session to a remote IMS system by using the ALLOCATE command.

The ALLOCATE command has the following format:

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

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

If CICS cannot find the named system, or no sessions are available, it raises the SYSIDERR condition. If CICS cannot find the named session or the session is out of service, CICS raises the SESSIONERR condition.

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

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

You can use the NOQUEUE option to specify explicitly that you do not want your request for a session to be queued if a session is not available immediately. A session is "not immediately available" in any of the following situations:

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

The action taken by CICS if a session is not immediately available depends on whether you specify NOQUEUE and also on whether your application has issued a HANDLE (that is still active) for the SYSBUSY condition. The possible combinations are as follows:

- Active HANDLE for SYSBUSY condition.

  Control is returned immediately to the label specified in the HANDLE command, regardless of whether you specified NOQUEUE.
- No active HANDLE for SYSBUSY condition.
  - If you specified NOQUEUE, control is returned immediately to your application program. The SYSBUSY code (X'D3') is set in the EIBRCODE field of the EXEC interface block. You should test this field immediately after issuing the ALLOCATE command.
  - If you omitted the NOQUEUE option, CICS queues the request until a session is available.

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

Similar considerations apply to an ALLOCATE command that specifies SESSION rather than SYSID. The associated condition is SESSBUSY (EIBRCODE= X'D2').

The session identifier

When a session has been allocated, the name by which it is known is available in the EIBRSRCE field in the EIB.

Because EIBRSRCE will probably be overwritten by the next EXEC CICS command, you must acquire the session name immediately. It is the name that you must use in the SESSION parameter of all subsequent commands that relate to this session.

Automatic transaction initiation

If the front-end transaction is designed to be started by automatic transaction initiation (ATI) in the local system, and is required to hold a conversation with an LUTYPE6.1 session as its principal facility, the session has already been allocated when the transaction starts.

You can omit the SESSION parameter from commands that relate to the principal facility. If, however, you want to name the session explicitly in these commands, you should obtain the name from EIBTRMID.

Attaching the remote transaction

When a session has been acquired, the next step is to cause the remote IMS process to be initiated.

The LUTYPE6.1 architecture defines a special function management header, called an attach header, which carries the name of the remote process (in CICS terms, the transaction) that is to be initiated, and also contains further session-related information.

CICS provides the BUILD ATTACH command to enable a CICS application program to build an attach header to send to IMS, and the EXTRACT ATTACH command to enable information to be obtained from attach headers received from IMS.

Because these commands are available, you do not need to know the detailed format of an LUTYPE6.1 attach header. In most cases, however, you need to know the meaning of the information that it carries.

The format of the BUILD ATTACH command is:

```
 BUILD ATTACH
ATTACHID(name)
[PROCESS(ISCEDT|BASICEDTvname)]
[RESOURCE(name)]
[RPROCESS(name)]
[RRESOURCE(name)]
[QUEUE(name)]
[IUTYPE(0|data-value)]
[DATASTR(0|data-value)]
[RECFM(data-value)]
```

The parameters of the BUILD ATTACH command have the following meanings:

ATTACHID(name)

The ATTACHID option enables you to assign a name to the attach header so that you can refer to it in a subsequent SEND or CONVERSE command. (The BUILD ATTACH command builds an attach header; it does not transmit it.)

PROCESS(name)

This corresponds to the process name, ATTDPN, in an attach FMH. It specifies the remote process that is to be initiated.

In CICS-to-IMS communication, the remote process is always an editor. It can be ISCEDT (or its alias), BASICEDT, or an MFS MID name. The process name must not exceed eight characters.

If the PROCESS option is omitted, IMS assumes ISCEDT.

RESOURCE(name)

This corresponds to the resource name, ATTPRN, in an attach FMH.

The RESOURCE option specifies the primary resource name (up to eight characters) that is to be assigned to the remote process that is being initiated.

In CICS-to-IMS communication, the primary resource name is either an IMS transaction code or a logical terminal name. You can omit the RESOURCE option if the IMS message destination is specified in the first eight bytes of the message or if the destination is preset by the IMS operator.

If a primary resource name is supplied to IMS, the data stream is not edited for destination and security information. You should therefore omit the RESOURCE option if IMS password processing is required.

The name in the RESOURCE option is ignored during conversational processing, or if the remote process is BASICEDT.

RPROCESS(name)

This corresponds to the return process name, ATTRDPN, in an attach FMH.

The RPROCESS option specifies a suggested return destination process name. IMS returns this name as a destination process name (ATTDPN) when it sends a reply to CICS, although the name may be overridden by MFS.

CICS uses the returned destination process name to determine the transaction that is to be attached after a session restart. At any other time, it is ignored. The RPROCESS option should therefore name a transaction that will handle any queued messages when it is attached by CICS at session restart following a session failure.

RRESOURCE(name)

This corresponds to the return resource name, ATTRPRN, in an attach FMH.

The RRESOURCE option specifies a suggested primary resource name that is to be assigned to the return process. IMS returns this name as the resource name (ATTPRN) when it sends a reply to CICS.

Although CICS normally ignores this field, one use for it in ISC is to specify a CICS terminal to which output messages occurring after session restart should be sent.

QUEUE(name)

This corresponds to the queue name, ATTDQN, in an attach FMH.

The QUEUE option specifies a queue that can be associated with the remote process. In CICS-to-IMS communication, it is used only to send a paging request to IMS during demand paging. The name used must be the one obtained by a previous EXTRACT ATTACH QNAME command. The name must not exceed eight characters.

IUTYPE(data-value)

This corresponds to the interchange unit field, ATTIU, in an attach FMH.

The IUTYPE option specifies SNA chaining information for the message. The value is halfword binary. The bits in the binary value are used as follows:

0–7	X'00' – must be set to zero

8–15	X'00' – multiple RU chains
	X'01' – single RU chains.

DATASTR(data-value)

This corresponds to the data stream profile field, ATTDSP, in an attach FMH.

The DATASTR option is used to select an IMS component. The value is halfword binary. The bits in the binary value are used as follows:

0–7	X'00' – must be set to zero
8–11	0000 – (user-defined data stream)
12–15	0000 – IMS Component 1
	0001 – IMS Component 2
	0010 – IMS Component 3
	0011 – IMS Component 4.

If the DATASTR option is omitted, IMS Component 1 is assumed.

RECFM(data-value)

This corresponds to the deblocking algorithm field, ATTDBA, in an attach FMH.

The RECFM option specifies the format of the user data that is sent to the remote process. The name must represent a halfword binary value. The bits in the binary value are used as follows:

0–7	X'00' – reserved – must be set to zero
8–15	X'01' – variable-length variable-blocked (VLVB) format
	X'04' – chain of RUs.

If VLVB is specified, your application program must add a two-byte binary length field in front of each record. If chain of RUs is specified, you can send your data in the usual way; no length fields are required.

A record is interpreted by IMS as either a segment of a message (without MFS) or an MFS record (with MFS).

The RECFM option indicates only the type of the message format. Multiple records can be sent by one SEND command. In this case, it is the responsibility of your application program to perform the blocking.

Having built the attach header, you must ensure that it is transmitted with the first data sent to the remote system by naming it in the ATTACHID option of the SEND or CONVERSE command.

Building your own attach header

CICS allows you to build an attach header, or any function management header, as part of your output data.

You can therefore initiate the remote transaction by including an LUTYPE6.1 attach header in the output area referenced by the first SEND or CONVERSE command. You must specify the FMH option on the command to tell CICS that the data contains an FMH.

Considerations for the back-end transaction

A CICS transaction can be the back-end transaction in CICS-to-IMS communication only in the special case of SEND and RECEIVE asynchronous processing.

The transaction is initiated by an LUTYPE6.1 attach FMH received from the remote IMS system, and is allowed to issue only a single RECEIVE command, possibly followed by an EXTRACT ATTACH command.

Acquiring session-related information

You can use the EXTRACT ATTACH command to recover session-related information from the attach FMH if required, but the use of this command is not mandatory.

The presence of an attach header is indicated by EIBATT, which is set after the first RECEIVE command has been issued.

The EXTRACT ATTACH command has the following format:

```
 EXTRACT ATTACH
[SESSION(data-area)]
[PROCESS(data-area)]
[RESOURCE(data-area)]
[RPROCESS(data-area)]
[RRESOURCE(data-area)]
[QUEUE(data-area)]
[IUTYPE(data-area)]
[DATASTR(data-area)]
[RECFM(data-area)]
```

The parameters of the EXTRACT ATTACH command have the following meanings:

DATASTR(data-area)

Contains a value specifying the IMS output component.

The data area must be a halfword binary field. The values set by IMS are as follows:

0–7	X'00' – (zero)
8–11	0000 – (user-defined data stream)
12–15	0000 – IMS Component 1
	0001 – IMS Component 2
	0010 – IMS Component 3
	0011 – IMS Component 4.

IUTYPE(data-area)

indicates SNA chaining information for the message and the type of MFS paged output.

The data area must be a halfword binary field. The values set by IMS are as follows:

0–7	X'00' – (zero)
8–15	X'00' – multiple RU chains, MFS autopaged output
	X'01' – single RU chains, MFS nonpaged output
	X'05' – single RU chains, MFS demand-paged output.

PROCESS(data-area)

IMS returns either the return destination process name specified in the RPROCESS option of the BUILD ATTACH command, or a value set by the MFS MOD.

QUEUE(data-area)

IMS returns the LTERM name associated with the ISC session when MFS demand-paged output is ready to be sent. The returned value should be used in the QMODEL FMH and the BUILD ATTACH QNAME when a paging request is to be sent.

RECFM(data-area)

Contains the data format of the incoming user message.

The data area must be a halfword binary field. The values set by IMS are as follows:

0–7	X'00' – (zero)
8–15	X'01' – variable-length variable-blocked (VLVB) format

X'04' – chain of RUs (can also be X'00' or X'05').

If VLVB is specified, your application program must deblock the message by using the halfword-binary length field that precedes each record.

RESOURCE(data-area)
IMS returns either the return resource name specified in the RRESOURCE option of the BUILD ATTACH command, or a value set by the MFS MOD.

RPROCESS(data-area)
IMS sends the chained MFS MID name if MFS is being used. Otherwise, no value is sent.

RRESOURCE(data-area)
IMS sends the value set by the MFS MOD if MFS is being used. Otherwise, no value is sent.

Initial state of back-end transaction

The back-end transaction is initiated in receive state, and should issue RECEIVE as its first command or after EXTRACT ATTACH.

The conversation

The conversation between the front-end and the back-end transactions is held using the usual SEND, RECEIVE, and CONVERSE commands.

For programming information about these commands, see SEND (LUTYPE6.1) , RECEIVE (LUTYPE6.1) , and CONVERSE (LUTYPE6.1).

In each of these comman ds, you must name the session in the SESSION option unless the conversation is with the principal facility.

Deferred transmission

On ISC sessions, when you issue a SEND command, CICS normally defers sending the data until it becomes clear what your further intentions are. This mechanism enables CICS to avoid unnecessary flows by adding control indicators on the data that is awaiting transmission.

In general, IMS does not accept indicators such as change-direction, syncpoint-request, or end-bracket as stand-alone transmissions on null RUs. You should therefore always allow deferred transmission to operate, and avoid using the WAIT option or the WAIT TERMINAL command to force transmissions to take place.

Using the LAST option

The LAST option on the SEND command indicates the end of the conversation. No further data flows can occur on the session, and the next action must be to free the session. However, the session can still carry CICS syncpointing flows before it is freed.

The LAST option and syncpoint flows

A syncpoint on an ISC session is initiated explicitly by a SYNCPOINT command, or implicitly by a RETURN command.

If your conversation has been terminated by a SEND LAST command, without the WAIT option, transmission has been deferred, and the syncpointing activity causes the final transmission to occur with an added syncpoint request. The conversation is thus automatically involved in the syncpoint.

Freeing the session

You must free the session after issuing a SEND LAST command, or when the EIBFREE field has been set.

The command used to free the session has the following format:

```
FREE SESSION(conversation-name)
```

CICS allows you to issue the FREE command at any time that your transaction is in send state. CICS determines whether the end-bracket indicator has already been transmitted, and transmits it if necessary before freeing the session. If there is also deferred data to transmit, the end-bracket indicator is transmitted with the data. Otherwise, the indicator is transmitted by itself.

Because only some IMS input components accept a stand-alone end-bracket indicator, this use of FREE is not recommended for CICS-to-IMS communication.

The EXEC interface block (EIB)

The EIB fields that are of particular significance in ISC applications are described.

For programming information about the EXEC interface block (EIB), see EIB fields . For further details of how and when to test or save these fields, see "Command sequences for CICS-to-IMS sessions" on page 281.

Conversation identifier fields

The EIB fields EIBTRMID and EIBRSRCE enable you to obtain the name of the ISC session.

EIBTRMID
Contains the name of the principal facility. For a back-end transaction, or for a front-end transaction started by ATI, it is the conversation identifier (SESSION). You must acquire this name if you want to state the session name of the principal facility explicitly.

EIBRSRCE
Contains the session identifier (SESSION) for the session obtained by means of an ALLOCATE command. You must acquire this name immediately after issuing the ALLOCATE command.

Procedural fields

These fields contain information on the state of the session. In most cases, the settings relate to the session named in the last-executed RECEIVE or CONVERSE command, and should be tested, or saved for later testing, after the command has been issued.

Further information about the use of these fields is given in "Command sequences for CICS-to-IMS sessions" on page 281.

EIBRECV
Indicates that the conversation is in receive state and that the normal continuation is to issue a RECEIVE command.

EIBCOMPL
This field is used in conjunction with the RECEIVE NOTRUNCATE command; it is set when there is no more data available.

EIBSYNC
Indicates that the application must take a syncpoint or terminate.

EIBSIG
Indicates that the conversation partner has issued an ISSUE SIGNAL command.

EIBFREE
Indicates that the receiver must issue a FREE command for the session.

Information fields

The following fields contain information about FMHs received from the remote transaction.

EIBATT
Indicates that the data received contained an attach header. The attach header is not passed to your application program; however, EIBATT indicates that an EXTRACT ATTACH command is appropriate.

EIBFMH
Indicates that the data passed to your application program contains a concatenated FMH.

If you want to use these facilities, you must ensure that you use communication profiles that specify INBFMH(ALL). The default profile (DFHCICSA) for a session allocated by a CICS front-end transaction has this specification. However, the default principal facility profile (DFHCICST) for a CICS back-end transaction does not. Further information about this subject is given under Defining communication profiles.

Command sequences for CICS-to-IMS sessions

The command sequences that you use to communicate between the front-end and the back-end transactions are governed both by the requirements of your application and by a set of high-level protocols designed to ensure that commands are not issued in inappropriate circumstances.

The protocols presented in this section do not cover all possible command sequences. However, by following them, you ensure that each transaction takes account of the requirements of the other. This helps to avoid errors during program development.

Conversation states

The protocols are based on the concept of several separate states.

These states apply only to the particular conversation, not to your entire application program. In each state, there is a choice of commands that might most reasonably be issued. After the command has been issued, fields in the EIB can be tested to learn the current requirements of the conversation. The results of these tests, together with the command that has been issued, may cause a transition to another state, when another set of commands becomes appropriate.

The states that are defined for this section are:

- State 1 – Session not allocated
- State 2 – Send state
- State 3 – Receive pending after SEND INVITE
- State 4 – Receive state
- State 5 – Receiver take syncpoint
- State 6 – Free pending after SEND LAST
- State 7 – Free session.

Initial states

Normally, the front-end transaction in a conversation starts in state 1 (session not allocated) and must issue an ALLOCATE command to acquire a session.

An exception to this occurs when the front-end transaction is started by automatic transaction initiation (ATI), in the local system, with an LUTYPE6.1 session as its principal facility. Here, the session is already allocated, and the transaction is in state 2. For transactions of this type, you must immediately obtain the session name from EIBTRMID so that you can name the session explicitly on later commands.

You must always assume that the back-end transaction is initially in state 4 (receive state). Even if it is designed only to send data to the front-end transaction, you must issue a RECEIVE to receive the SEND INVITE issued by the front-end transaction and get into send state.

State diagrams

You can use the state diagrams to construct valid command sequences. Each diagram relates to one specific state and shows the commands that you might issue, and the tests to make, after you issue the command.

When a diagram shows more than one test, make them in the order indicated. The combination of the command issued and a particular positive test result leads to a new resultant state, shown in the final column.

The state diagrams shows tests that are significant to the state of the conversation. For other conditions that can occur, for example, INVREQ or NOTALLOC, make tests in the normal way.

Table 24. State 1: CICS-TO-IMS conversations - session not allocated

Commands you can issue	What to test	New state
ALLOCATE [NOQUEUE] *	SYSIDERR	1
Ditto	SYSBUSY *	1
Ditto	Otherwise (obtain session name from EIBRSRCE)	2

If you want your program to wait until a session is available, omit the NOQUEUE option of the ALLOCATE command and do not code a HANDLE command for the SYSBUSY condition.

If you want control to be returned to your program if a session is not immediately available, either specify NOQUEUE on the ALLOCATE command and test EIBRCODE for SYSBUSY (X'D3'), or code a HANDLE CONDITION SYSBUSY command.

Table 25. State 2: CICS-TO-IMS conversations - send state

Commands you can issue *	What to test	New state
SEND		2
SEND INVITE	—	3 or 4
SEND LAST	—	6

CONVERSE
`Equivalent to:`
`SEND INVITE WAIT`
`RECEIVE`
	Go to the STATE 4 table and make the tests shown for the RECEIVE command.	—

RECEIVE	Go to the STATE 4 table and make the tests shown for the RECEIVE command.	—
SYNCPOINT	(Transaction abends if SYNCPOINT fails.)	2

FREE
`Equivalent to:`
`SEND LAST WAIT`
`FREE`
	—	1

For the front-end transaction, the first command used after the session has been allocated must be a SEND command or CONVERSE command that initiates the back-end transaction in one of the ways described under "Attaching the remote transaction" on page 275.

Table 26. State 3: CICS-TO-IMS conversations - receive pending after SEND INVITE

Commands you can issue	What to test	New state
SYNCPOINT	(Transaction abends if SYNCPOINT fails.)	4

Table 27. State 4: CICS-TO-IMS conversations - receive state

Commands you can issue	What to test	New state
RECEIVE [NOTRUNCATE] *	EIBCOMPL *	—
Ditto	EIBSYNC	5
Ditto	EIBFREE	7
Ditto	EIBRECV	4

Table 27. State 4: CICS-TO-IMS conversations - receive state (continued)

Commands you can issue	What to test	New state
Ditto	Otherwise	2

If NOTRUNCATE is specified, a zero value in EIBCOMPL indicates that the data passed to the application by CICS is incomplete (because, for example, the data area specified in the RECEIVE command is too small). CICS saves the remaining data for retrieval by later RECEIVE NOTRUNCATE commands. EIBCOMPL is set when the last part of the data is passed back. If the NOTRUNCATE option is not specified, over-length data is indicated by the LENGERR condition, and the remaining data is discarded by CICS.

Table 28. State 5: CICS-TO-IMS conversations - receiver take syncpoint

Commands you can issue	What to test	New state
SYNCPOINT	EIBFREE (saved value)	7
Ditto	EIBRECV (saved value)	4
Ditto	Otherwise	2

Table 29. State 6: CICS-TO-IMS conversations - free pending after SEND LAST

Commands you can issue	What to test	New state
SYNCPOINT	—	7
FREE	—	1

Table 30. State 7: CICS-TO-IMS conversations - free session

Commands you can issue	What to test	New state
FREE	—	1

Chapter 6. Improving intersystem performance

There are a number of ways that you can improve aspects of CICS performance in a multi-system environment.

"Intersystem session queue management" on page 285 describes methods for controlling the length of intersystem queues.

"Efficient deletion of shipped terminal definitions" on page 287 describes how to delete redundant shipped terminal definitions from AORs and intermediate systems.

Intersystem session queue management

This section describes how to control the number of queued requests for sessions on intersystem links (allocate queues).

Note: This section describes how to control queues for sessions on established connections. The specialized subject of using local queuing for function-shipped **EXEC CICS START NOCHECK** requests is described in Local queuing of START commands.

Overview of session queue management

In a perfect intercommunication environment, queues would never occur because work flow would be evenly distributed over time, and there would be enough intersystem sessions available to handle the maximum number of requests arriving at any one time.

However, in the real world this is not the case, and, with peaks and troughs in the workload, queues do occur: queues come and go in response to the workload. The situation to avoid is an unacceptably high level of queuing that causes a bottleneck in the work flow between interconnected CICS regions, and which leads to performance problems for the terminal end-user as throughput slows down or stops. This abnormal and unexpected queuing should be prevented, or dealt with when it occurs: a "normal" or optimized level of queuing can be tolerated.

For example, function shipping requests between CICS application-owning regions and connected file-owning regions can be queued in the issuing region while waiting for free sessions. Provided a file-owning region deals with requests in a responsive manner, and outstanding requests are removed from the queue at an acceptable rate, then all is well. But if a file-owning region is unresponsive, the queue can become so long and occupy so much storage that the performance of connected application-owning regions is severely impaired. Further, the impaired performance of the application-owning region can spread to other regions. This condition is sometimes referred to as "sympathy sickness", although it should more properly be described as intersystem queuing, which, if not controlled, can lead to performance degradation across more than one region.

How to manage allocate queues

There are three methods for managing allocate queues.

Using resource definitions to manage allocate queues

You can specify the QUEUELIMIT and MAXQTIME options on the CONNECTION and IPCONN resource definitions for intersystem links that have simple control requirements; for example, links that carry noncritical traffic.

QUEUELIMIT defines the maximum number of allocate requests that CICS is to queue while waiting for free sessions on the connection.

MAXQTIME defines the approximate time for which allocate requests will queue for free sessions on a connection that is unresponsive. MAXQTIME is used only if a queue limit is specified on QUEUELIMIT, and if that limit is reached.

When an allocate request is received that causes the QUEUELIMIT value to be exceeded, CICS calculates whether the rate of processing of the queue will allow the new request to be processed in the maximum queuing time. If the request is not processed, CICS purges the queue. No further queuing takes place until the connection has freed a session. At this point, queuing begins again.

When CICS purges an allocate request because the QUEUELIMIT and MAXQTIME settings are exceeded, the SYSIDERR condition is returned to the application program.

For information about the QUEUELIMIT and MAXQTIME attributes, see CONNECTION attributes and IPCONN attributes.

Using the NOQUEUE option to manage allocate queues

A further method of controlling *explicit* allocate requests is to specify the NOQUEUE|NOSUSPEND option of the **`EXEC CICS ALLOCATE`** command.

However, while this enables you to control specific requests, it takes no account of the state of the queue at the time the requests are issued. And it is of no use in controlling *implicit* allocate requests (where the session request is instigated by, for example, a function shipping request). For programming information about API options, see ALLOCATE (APPC) .

Using the XISQUE and XZIQUE global user exits to manage allocate queues

You can control the queuing of allocate requests through a global user exit program, which provides more flexibility than setting a queue limit on the connection. Use XISQUE to manage IPIC queues and XZIQUE to manage MRO and APPC queues.

With the XISQUE and XZIQUE exits, you can quickly detect queuing problems (bottlenecks). Both exits enable allocate requests to be queued or rejected, depending on the length of the queue. You can use XISQUE and XZIQUE to stop and then reestablish a connection that has a bottleneck.

The XZIQUE exit extends the function that the XISCONA exit provides for MRO and APPC connections. XISCONA is called for function shipping and DPL requests only, including function shipped EXEC CICS START requests used for asynchronous processing. XZIQUE is called for transaction routing, asynchronous processing, and distributed transaction processing requests, in addition to function shipping and DPL. Compared with the XISCONA exit, XZIQUE receives more detailed information on which to base its action. For information about the relationship between XISCONA and XZIQUE, see Interaction with the XISCONA exit.

Uses of a queuing global user exit program

When the exit is enabled, your XZIQUE or XISQUE global user exit program can check on the state of the allocate queue for a particular connection in the local system.

Information is passed to the exit program in a parameter list that is structured to provide data about nonspecific allocate requests or requests for specific modegroups, depending on the session request. If you are using the XZIQUE exit, nonspecific allocate requests are for MRO, LU6.1, and APPC sessions that do not specify a modegroup.

Using the information passed in the parameter list, your global user exit program selects the system action to take:

- Queue the allocate request. This action is possible only if the queue limit has not been reached.
- Reject the allocate request.
- Reject this allocate request and purge all queued requests for the connection.
- Reject this allocate request and purge all queued requests for the modegroup.

Your exit program might base its action on one of the following criteria: Exit XISQUE

- The length of the allocate queue.
- Whether the number of queued requests has reached the limit set by the QUEUELIMIT option. If the queue limit has not been reached, you might decide to queue the request.
- The rate at which sessions are being allocated on the connection. If the queue limit has been reached but session allocation is acceptably quick, you might decide to reject only the current request. If the queue limit has been reached and session allocation is unacceptably slow, you might decide to purge the whole queue.

For details of the information passed in the XISQUE parameter list, and advice about designing and coding an XISQUE exit program, see the programming information in XISQUE exit for managing IPIC intersystem queues.

For details of the information passed in the XZIQUE parameter list, and advice about designing and coding an XZIQUE exit program, see the programming information in XZIQUE exit for managing MRO and APPC intersystem queues.

Efficient deletion of shipped terminal definitions

This section describes how CICS deletes redundant shipped terminal definitions.

Overview of how shipped terminals are deleted

In a transaction routing environment, terminal definitions can be shipped from a terminal-owning region (TOR) to an application-owning region (AOR) when they are first needed, rather than being statically defined in the AOR.

The terminal could be an APPC device or system. In this case, the shipped definition would be of an APPC connection.

Shipped definitions can become redundant in the following situations:

- A terminal user logs off.
- A terminal user stops using remote transactions.
- The TOR is shut down.
- The TOR is restarted, autoinstalled terminal definitions are not recovered, and the autoinstall user program, DFHZATDX, assigns a new set of termids to the same set of terminals.

At some stage, redundant definitions must be deleted from the AOR (and from any intermediate systems between the TOR and AOR). For brevity, AORs and intermediate systems are collectively referred to as *back-end systems*. This is particularly necessary in the last situation in the previous list, to prevent a possible mismatch between termids in the TOR and the back-end systems.

CICS method of deleting redundant shipped definitions consists of two parts:

Selective deletion

Each time a terminal definition is installed, CICS creates a unique “instance token” and stores it within the definition.

Thus, if the definition is shipped to another region, the value of the token is shipped too. All transaction routing attach requests pass the token within the function management header (FMH). If, during attach processing, an existing shipped definition is found in the remote region, it is used *only if the token in the shipped definition matches that passed by the TOR*. Otherwise, it is deleted and an up-to-date definition shipped.

A timeout delete mechanism

You can use the timeout delete mechanism in your back-end systems, to delete shipped definitions that have not been used for transaction routing for a defined period Its purpose is to ensure that shipped definitions remain installed only while they are in use.

Note: Shipped definitions are not deleted if there is an automatic initiate descriptor (AID) associated with the terminal.

Timeout delete gives you flexible control over shipped definitions. CICS allows you to:

- Stipulate the minimum time a shipped definition must remain installed before being eligible for deletion
- Stipulate the time interval between invocations of the mechanism
- Reset these times online
- Cause the timeout delete mechanism to be invoked immediately.

The parameters that control the mechanism allow you to arrange for a tidy-up operation to take place when the system is least busy.

Implementing timeout delete

To use timeout delete in a CICS Transaction Server for z/OS system to which terminals are shipped, you specify two system initialization parameters.

DSHIPIDL={020000|hhmmss}

Specifies the minimum time, in hours, minutes, and seconds, that an *inactive* shipped terminal definition must remain installed in this region. When the CICS timeout delete mechanism is invoked, only those shipped definitions that have been inactive for longer than the specified time are deleted.

You can use this parameter in a transaction routing environment, on the application-owning and intermediate regions, to prevent terminal definitions having to be reshipped because they have been deleted prematurely.

hhmmss

Specify a 1 to 6 digit number in the range 0-995959. Numbers that have fewer than six digits are padded with leading zeros.

DSHIPINT={120000|0|hhmmss}

Specifies the interval between invocations of the CICS timeout delete mechanism. The timeout delete mechanism removes any shipped terminal definitions that have not been used for longer than the time specified by the DSHIPIDL parameter.

You can use this parameter in a transaction routing environment, on the application-owning and intermediate regions, to control:

- How often the timeout delete mechanism is invoked.
- The approximate time of day at which a mass delete operation is to take place, relative to CICS startup.

0

The timeout delete mechanism is not invoked. You might set this value in a terminal-owning region, or if you are not using shipped definitions.

hhmmss

Specify a 1 to 6 digit number in the range 1-995959. Numbers that have fewer than six digits are padded with leading zeros.

For details of how to specify system initialization parameters, see CICS system initialization.

After CICS startup you can examine the current settings of DSHIPIDL and DSHIPINT. For flexible control over when mass delete operations take place, you can reset the interval until the next invocation of the timeout delete mechanism. (The revised interval starts *from the time the command is issued*, **not** from the time the remote delete mechanism was last invoked, nor from CICS startup.) Alternatively, you can invoke the timeout delete mechanism.

Tuning the performance of timeout delete

A careful choice of DSHIPINT and DSHIPIDL settings results in a minimal number of mass deletions of shipped definitions, and a scheduling of those that do take place for times when your system is lightly loaded.

Conversely, a poor choice of settings could result in unnecessary mass delete operations. Here are some suggestions for coding DSHIPINT and DSHIPIDL:

DSHIPIDL

In setting this value, you must consider the length of the work periods during which remote users access resources on this system. Do they access the system intermittently, all day? Or is their work concentrated into intensive, shorter periods?

By setting too low a value, you could cause definitions to be deleted and reshipped unnecessarily. It is also possible that you could cause automatic transaction initiation (ATI) requests to fail with the "terminal not known" condition. This condition occurs when an ATI request names a terminal that is not defined to this system. Usually, the terminal is not defined because it is owned by a remote system, you are using shippable terminals, and no prior transaction routing has taken place from it. By allowing temporarily inactive shipped definitions too short a life, you could increase the number of calls to the XALTENF and XICTENF global user exits that deal with the "terminal not known" condition.

DSHIPINT

You can use this value to control the time of day at which your mass delete operations take place.

For example, if you usually warm-start CICS at 7 a.m., you could set DSHIPINT to 150000, so that the timeout delete mechanism is invoked at 10 p.m., when few users are accessing the system.

⚠ **Attention:** If CICS is recycled, perhaps because of a failure, the timeout delete interval is reset. Continuing the previous example, if CICS is recycled at 8:00 p.m., the timeout delete mechanism will be invoked at 11:00 a.m. the following day (15 hours from the time of CICS initialization). In these circumstances, you could use the SET DELETSHIPPED and PERFORM DELETSHIPPED commands to accurately control when a timeout delete takes place.

CICS provides statistics to help you tune the DFHIPIDL and DFHIPINT parameters. The statistics are available online, and are mapped by the DFHA04DS DSECT. For details of the statistics provided, see Autoinstall statistics.

Chapter 7. Troubleshooting intersystem problems

Messages that report CICS recovery actions and examples of how to resolve indoubt and resynchronization failures can help you troubleshoot problems between systems.

Messages that report CICS recovery actions

When a communications failure occurs, the connected systems might resolve their local parts of a distributed unit of work in ways that are inconsistent with each other. To warn of this possibility, when a CICS region loses communication with a partner, for each session on which the UOW is in the indoubt period, it issues a DFHRM*xxxx* message.

The message can be issued at the time of a session failure, a failure of the partner, or during emergency restart.

When the connection has been reestablished, on each affected session the UOW is unshunted, its state is determined, and another message is issued. For LUTYPE6.1 conversations, these messages might appear only on the initiator side.

All messages contain the following information, so they can be correlated:

- The time and date of the original failure
- The transaction identifier and task number
- The netname of the remote system
- The operator identifier
- The operator terminal identifier
- The network-wide unit of work identifier
- The local unit of work identifier.

The following types of messages associated with intersystem session failure and recovery are produced:

- Messages when contact is lost with the coordinator of the UOW. See Table 31 on page 291 and Table 32 on page 292.
- Messages when WAIT(YES) is specified on the transaction definition and shunting is possible. See Table 31 on page 291.
- Messages when WAIT(NO) is specified, or when shunting is not possible. See Table 32 on page 292.
- Messages when contact is lost with a subordinate in the UOW. See Table 33 on page 293.

For full details about a message, see CICS messages.

Table 31. WAIT(YES) session failure messages. The failure is between the session and the coordinator of the UOW, WAIT(YES) is specified on the transaction definition, and shunting is possible. In each stage (1 and 2), the messages that CICS issues depend on the circumstances that apply, as shown in columns 2 and 4. Stage 1 applies to MRO messages, and Stage 2 applies to IPIC and APPC messages.

Sequence of messages	Circumstances	Messages issued	Meaning of messages
Stage 1	Session failure	DFHRM0106	Intersystem session failure. Resource changes are not committed or backed out until session recovery.
Stage 1	System failure or restart	—	—
Stage 2	Session recovery successful	DFHRM0108	Intersystem session recovery. Suspended resource changes now being committed.

Table 31. WAIT(YES) session failure messages. The failure is between the session and the coordinator of the UOW, WAIT(YES) is specified on the transaction definition, and shunting is possible. In each stage (1 and 2), the messages that CICS issues depend on the circumstances that apply, as shown in columns 2 and 4. Stage 1 applies to MRO messages, and Stage 2 applies to IPIC and APPC messages. *(continued)*

Sequence of messages	Circumstances	Messages issued	Meaning of messages
Stage 2	Session recovery successful	DFHRM0109	Intersystem session recovery. Suspended resource changes now being backed out.
Stage 2	Wait time exceeded or SET UOW ACTION issued	DFHRM0104 DFHRM0105	See next table.
Stage 2	SET CONNECTION NOTPENDING or XLNACTION (FORCE) or NORECOVDATA issued	DFHRM0125 DFHRM0126	Local resources committed or backed out.
Stage 2	Session recovery after a cold start of local resources.	DFHRM0209	UOW backed out.
Stage 2	Session recovery after a cold start of local resources	DFHRM0208	UOW committed.
Stage 2	Session recovery error; for example, partner started cold "1" on page 292	DFHRM0112 DFHRM0113 DFHRM0115 DFHRM0116 DFHRM0118 DFHRM0119 DFHRM0121 DFHRM0122	Intersystem recovery error. Local resource changes are committed or backed out.

Note:

1. LU6.1 only

Table 32. WAIT(NO) session failure messages . The failure is between the session and the coordinator of the UOW. WAIT(NO) is specified on the transaction definition or shunting is not possible. In each stage (1 and 2), the messages that CICS issues depend on the circumstances that apply, as shown in columns 2 and 4. Stage 1 applies to MRO messages, and Stage 2 applies to IPIC and APPC messages.

Sequence of messages	Circumstances	Messages issued	Meaning of messages
Stage 1	Session failure	DFHRM0104 DFHRM0105	Intersystem session failure. Resource changes are being committed or backed out and might be out of sync with partner.
Stage 1	System failure or restart	—	—
Stage 2	Session recovery successful	DFHRM0110	Intersystem session recovery. Resource updates found to be synchronized.
Stage 2	Session recovery successful	DFHRM0111	Intersystem session recovery. Resource updates found to be out of sync.

Table 32. WAIT(NO) session failure messages . The failure is between the session and the coordinator of the UOW. WAIT(NO) is specified on the transaction definition or shunting is not possible. In each stage (1 and 2), the messages that CICS issues depend on the circumstances that apply, as shown in columns 2 and 4. Stage 1 applies to MRO messages, and Stage 2 applies to IPIC and APPC messages. *(continued)*

Sequence of messages	Circumstances	Messages issued	Meaning of messages
Stage 2	SET CONNECTION NOTPENDING or XLNACTION (FORCE) or NORECOVDATA issued	DFHRM0127	SET NOTPENDING issued.
Stage 2	Session recovery error; for example, partner started cold "1" on page 293	DFHRM0112 DFHRM0113 DFHRM0115 DFHRM0116 DFHRM0118 DFHRM0119 DFHRM0121 DFHRM0122	Local resource changes committed or backed out.

Note:

1. LU6.1 only

Table 33. Subordinate session failure messages. The failure is between the session and a subordinate in the UOW. In each stage (1 and 2), the messages that CICS issues depend on the circumstances that apply, as shown in columns 2 and 4. Stage 1 applies to MRO messages, and Stage 2 applies to IPIC and APPC messages.

Sequence of messages	Circumstances	Messages issued	Meaning of messages
Stage 1	UOW shunted because of failure of session to coordinator	—	—
Stage 1	Session failure	DFHRM0107	Intersystem session failure. Notification of decision might not reach the remote system.
Stage 1	System failure or restart	—	—
Stage 2	Session recovery successful	DFHRM0135 DFHRM0148 "1" on page 294	Intersystem session recovery. Resource updates found to be synchronized.
Stage 2	Session recovery successful	DFHRM0110	Intersystem session recovery. Resource updates found to be synchronized, after a unilateral decision on the remote system.
Stage 2	Session recovery successful	DFHRM0111 DFHRM0124	Intersystem session recovery. Resource updates found to be out of sync, after a unilateral decision on the remote system.

Table 33. Subordinate session failure messages. The failure is between the session and a subordinate in the UOW. In each stage (1 and 2), the messages that CICS issues depend on the circumstances that apply, as shown in columns 2 and 4. Stage 1 applies to MRO messages, and Stage 2 applies to IPIC and APPC messages. *(continued)*

Sequence of messages	Circumstances	Messages issued	Meaning of messages
Stage 2	SET CONNECTION NOTPENDING or XLNACTION (FORCE) or NORECOVDATA issued	DFHRM0127	SET NOTPENDING issued.
Stage 2	Session recovery error; for example, partner started cold "2" on page 294	DFHRM0114 DFHRM0117 DFHRM0120 DFHRM0123	Intersystem session recovery error. Resource changes might be out of sync.

Notes:

1. DFHRM0124 and DFHRM0148 might occur without a preceding session failure message (DFHRM0107) or shunt.
2. LU6.1 only

Problem determination examples

This section contains examples of how to resolve indoubt and resynchronization failures.

Resource definition

- The PRINTER and ALTPRINTER options for a z/OS Communications Server terminal must (if specified) name a printer owned by the same system as the one that owns the terminal being defined.
- The terminals listed in the terminal list table (DFHTLT) must reside on the same system as the terminal list table.

Resolving a resynchronization failure

An example of how to resolve a resynchronization failure by using the CEMT transaction is described.

This example uses the following commands:

- CEMT INQUIRE CONNECTION
- CEMT INQUIRE UOWLINK
- CEMT INQUIRE UOW
- CEMT INQUIRE UOWENQ
- SET CONNECTION NOTPENDING

Tip: In CICS Explorer, the **ISC/MRO Connections** view provides a functional equivalent to the INQUIRE and SET CONNECTION commands. See ISC/MRO Connections view in the CICS Explorer product documentation.

A transaction on system IYLX1 (which involves function shipping requests to system IYLX4) is failing with a SYSIDERR error. A CEMT INQUIRE CONNECTION command on system IYLX1 shows the following:

```
INQUIRE CONNECTION
STATUS:  RESULTS - OVERTYPE TO MODIFY
 Con(ISC2) Net(IYLX2   )     Ins Rel Vta Appc         Unk
 Con(ISC4) Net(IYLX4   ) Pen Ins Acq Vta Appc     Xno Unk
 Con(ISC5) Net(IYLX5   )     Ins Acq Vta Appc     Xok Unk
```

Figure 79. CEMT INQUIRE CONNECTION: connections owned by system IYLX1

To see more information about this connection, put the cursor on the ISC4 line and press ENTER. See Figure 80 on page 295.

```
INQUIRE CONNECTION
RESULT - OVERTYPE TO MODIFY
  Connection(ISC4)
  Netname(IYLX4)
  Pendstatus( Pending )
  Servstatus( Inservice )
  Connstatus( Acquired )
  Accessmethod(Vtam)
  Protocol(Appc)
  Purgetype(              )
  Xlnstatus(Xnotdone)
  Recovstatus( Nrs )
  Uowaction(              )
  Grname()
  Membername()
  Affinity(              )
  Remotesystem()
  Rname()
  Rnetname()
```

Figure 80. CEMT INQUIRE CONNECTION: details of connection ISC4

Note: VTAM is the previous name for z/OS Communications Server.

Although the Connstatus of connection ISC4 is **Acquired**, the Xlnstatus is **Xnotdone**. The exchange lognames (XLN) flow for this connection has not completed successfully. (When CICS systems connect they exchange lognames. These lognames are verified before resynchronization is attempted, and an exchange lognames failure means that resynchronization is not possible.) For function shipping, a failure for the connection causes a SYSIDERR. Synchronization level 2 conversations are not allowed on this connection until lognames are successfully exchanged. (This restriction does not apply to MRO connections.)

The reason for the exchange lognames failure is reported in the CSMT log. A failure on a CICS Transaction Server for z/OS system can be caused by the following:

- An initial start (START=INITIAL) of the CICS TS for z/OS system, or of a partner.

  Note: A cold start (START=COLD) of a CICS TS for z/OS system preserves resynchronization information (including the logname) and does not, therefore, cause an exchange lognames failure.
- Use of the CEMT SET CONNECTION NORECOVDATA command.
- A system logic or operational error.

The Pendstatus for connection ISC4 is **Pending**, which means that there is resynchronization work outstanding for the connection; this work cannot be completed because of the exchange lognames failure.

At this stage, if you are not concerned about loss of synchronization, you can force all indoubt UOWs to commit or back out by issuing the SET CONNECTION NOTPENDING command. However, before you do so, you can investigate the outstanding resynchronization work that exists before we clear the pending condition.

You can use a CEMT INQUIRE UOWLINK command to display information about UOWs that require resynchronization with system IYLX4:

```
INQUIRE UOWLINK LINK(IYLX4)
STATUS:  RESULTS - OVERTYPE TO MODIFY
 Uowl(016C0005) Uow(ABD40B40C1334401) Con Lin(IYLX4   )
    Coo Appc Col Sys(ISC4)            Net(..GBIBMIYA.IYLX150 M. A....)
 Uowl(01680005) Uow(ABD40B40C67C8201) Con Lin(IYLX4   )
    Coo Appc Col Sys(ISC4)            Net(..GBIBMIYA.IYLX151 M. F@b..)
 Uowl(016D0005) Uow(ABD40B40DA5A8803) Con Lin(IYLX4   )
    Coo Appc Col Sys(ISC4)            Net(..GBIBMIYA.IYLX156 M. .!h..)
```

Figure 81. CEMT INQUIRE UOWLINK: UOWs that require resynchronization with system IYLX4

To see more information for each UOW-link, press enter alongside it. For example, the expanded information for UOW-link 016C0005 shows the following:

```
I UOWLINK LINK(IYLX4)
RESULT - OVERTYPE TO MODIFY
  Uowlink(016C0005)
  Uow(ABD40B40C1334401)
  Type(Connection)
  Link(IYLX4)
  Action(              )
  Role(Coordinator)
  Protocol(Appc)
  Resyncstatus(Coldstart)
  Sysid(ISC4)
  Rmiqfy()
  Netuowid(..GBIBMIYA.IYLX150 M. A....)
```

Figure 82. CEMT INQUIRE UOWLINK: detailed information for UOW-link 016C0005

The Resyncstatus of **Coldstart** confirms that system IYLX4 has been started with a new logname. The Role for this UOW-link is shown as **Coordinator**, which means that IYLX4 is the syncpoint coordinator.

You could now use a CEMT INQUIRE UOW LINK(IYLX4) command to show all UOWs that are indoubt and which have system IYLX4 as the coordinator system:

```
INQUIRE UOW LINK(IYLX4)
STATUS:  RESULTS - OVERTYPE TO MODIFY
 Uow(ABD40B40C1334401) Ind Shu Tra(RFS1) Tas(0000674)
    Age(00003560) Ter(X150) Netn(IYLX150 ) Use(CICSUSER) Con Lin(IYLX4   )
 Uow(ABD40B40C67C8201) Ind Shu Tra(RFS1) Tas(0000675)
    Age(00003465) Ter(X151) Netn(IYLX151 ) Use(CICSUSER) Con Lin(IYLX4   )
 Uow(ABD40B40DA5A8803) Ind Shu Tra(RFS1) Tas(0000676)
    Age(00003462) Ter(X156) Netn(IYLX156 ) Use(CICSUSER) Con Lin(IYLX4   )
```

Figure 83. CEMT INQUIRE UOW LINK(IYLX4): all UOWs that have IYLX4 as the coordinator

To see more information for each indoubt UOW, press enter on its line. For example, the expanded information for UOW ABD40B40C1334401 shows the following:

```
INQUIRE UOW LINK(IYLX4)
RESULT - OVERTYPE TO MODIFY
  Uow(ABD40B40C1334401)
  Uowstate( Indoubt )
  Waitstate(Shunted)
  Transid(RFS1)
  Taskid(0000674)
  Age(00003906)
  Termid(X150)
  Netname(IYLX150)
  Userid(CICSUSER)
  Waitcause(Connection)
  Link(IYLX4)
  Sysid(ISC4)
  Netuowid(..GBIBMIYA.IYLX150 M. A....)
```

Figure 84. CEMT INQUIRE UOW LINK(IYLX4): detailed information for UOW ABD40B40C1334401

This UOW cannot be resynchronized by system IYLX4. Its status is shown as **Indoubt**, because IYLX4 does not know whether the associated UOW that ran on IYLX4 committed or backed out.

You can use the CEMT INQUIRE UOWENQ command to display the resources that have been locked by all shunted UOWs (those that own retained locks):

```
INQUIRE UOWENQ OWN RETAINED
STATUS:  RESULTS
 Uow(ABD40B40C1334401) Tra(RFS1) Tas(0000674) Ret Tsq Own
    Res(RFS1X150                  ) Rle(008) Enq(00000008)
 Uow(ABD40B40C67C8201) Tra(RFS1) Tas(0000675) Ret Tsq Own
    Res(RFS1X151                  ) Rle(008) Enq(00000008)
 Uow(ABD40B40DA5A8803) Tra(RFS1) Tas(0000676) Ret Tsq Own
    Res(RFS1X156                  ) Rle(008) Enq(00000008)
```

Figure 85. CEMT INQUIRE UOWENQ: resources locked by all shunted UOWs

You can filter the INQUIRE UOWENQ command so that only enqueues that are owned by a particular UOW are displayed. For example, to filter for enqueues owned by UOW ABD40B40C1334401:

```
INQUIRE UOWENQ OWN UOW(*4401)
STATUS:  RESULTS
 Uow(ABD40B40C1334401) Tra(RFS1) Tas(0000674) Ret Tsq Own
    Res(RFS1X150                  ) Rle(008) Enq(00000008)
```

Figure 86. CEMT INQUIRE UOWENQ: resources locked by UOW ABD40B40C1334401

To see more information for this UOWENQ, press enter alongside it:

```
INQUIRE UOWENQ OWN UOW(*4401)
RESULT
  Uowenq
  Uow(ABD40B40C1334401)
  Transid(RFS1)
  Taskid(0000674)
  State(Retained)
  Type(Tsq)
  Relation(Owner)
  Resource(RFS1X150)
  Rlen(008)
  Enqfails(00000008)
  Netuowid(..GBIBMIYA.IYLX150 M. A....)
  Qualifier()
  Qlen(000)
```

Figure 87. CEMT INQUIRE UOWENQ: detailed information for UOWENQ ABD40B40C1334401

With knowledge of the application, it may now be possible to decide whether updates to the locked resources should be committed or backed out. In the case of UOW ABD40B40C1334401, the locked resource is the temporary storage queue RFS1X150. This resource has an ENQFAILS value of 8, which is the number of tasks that have received the **LOCKED** response due to this enqueue being held in retained state.

You can use the SET UOW command to commit, back out, or force the uncommitted updates made by the shunted UOWs. Next, you must use the SET CONNECTION(ISC4) NOTPENDING command to clear the pending condition and allow synchronization level 2 conversations (including the function shipping requests which were previously failing with SYSIDERR).

You can use the XLNACTION option of the CONNECTION definition to control the effect of an exchange lognames failure. In this example, the XLNACTION for the connection ISC4 is **KEEP**. This meant that:

- The shunted UOWs on system IYLX1 were kept following the cold/warm log mismatch with IYLX4.
- The APPC connection between IYLX1 and IYLX4 could not be used for function shipping requests until the pending condition was resolved.

An XLNACTION of **FORCE** for connection ISC4 would have caused the SET CONNECTION NOTPENDING command to have been issued automatically when the cold/warm log mismatch occurred. This would have forced the shunted UOWs to commit or back out, according to the ACTION option of the associated transaction definition. The connection ISC4 would then not have been placed into **Pending** status. However, setting XLNACTION to FORCE allows no investigation of shunted UOWs following an exchange lognames failure, and therefore represents a greater risk to data integrity than setting XLNACTION to KEEP.

APPC connection quiesce processing

When an APPC parallel-session connection with a CICS Transaction Server for z/OS region is shut down normally, CICS exchanges information with its partner to discover if there is any possibility that resynchronization is required when the connection is restarted.

This exchange is known as the connection quiesce protocol (CQP).

CICS determines that resynchronization is not required if all the following conditions are true:

- The connection is being shut down.
- There are no user sessions active (the CQP uses the SNASVCMG sessions). If the SNASVCMG sessions become inactive before the user sessions, the CQP will not take place.
- The CICS recovery manager domain has no record of outstanding syncpoint work or resynchronization work for the connection.

Once the CQP has completed, CICS ensures that no recoverable work can be initiated for the connection until a fresh logname exchange has taken place.

If the CQP determines that resynchronization is not required, CICS:

- Sets the connection's recovery state to NORECOVDATA
- If CICS is a member of a generic resource group, ends any affinity held by z/OS Communications Server and issues a message to say that the affinity has been ended.

If there is any failure of the CQP, CICS presumes that there is a possibility of resynchronization being necessary. You may use the procedures described here to determine if this is truly the case, and perform the necessary actions manually. Alternatively, you can reacquire the connection and release it again, to force CICS to re-attempt the CQP.

Syncpoint exchanges

Consider the following example:

Syncpoint example:

An order-entry transaction is designed so that, when an order for an item is entered from a terminal:

1. ***An inventory file is queried and decremented by the order quantity.***
2. ***An order for dispatch of the goods is written to an intrapartition transient data queue.***
3. ***A synchronization point is taken to indicate the end of the current UOW.***

In a single CICS system, the syncpoint causes steps “1” on page 298 and “2” on page 298 both to be committed.

The same result is required if the inventory file is owned by a remote system and is accessed by means of, for example, CICS function shipping. This is achieved in the following way:

1. When the local transaction issues the syncpoint request, CICS sends a syncpoint request to the remote transaction (in this case, the CICS mirror transaction).
2. The remote transaction commits the change to the inventory file and sends a positive response to the local CICS system.
3. CICS commits the change to the transient data queue.

During the period between the sending of the syncpoint request to the remote system and the receipt of the reply, the local system does not know whether the remote system has committed the change. This period is known as the **indoubt** period, as illustrated in Figure 88 on page 299.

If the intersystem session fails before the indoubt period is reached, both sides back out in the normal way. After this period, both sides have committed their changes. If, however, the intersystem session fails during the indoubt period, the local CICS system cannot tell whether the remote system committed or backed out its changes.

Syncpoint flows

The ways in which syncpoint requests and responses are exchanged on intersystem conversations are defined in the APPC and LUTYPE6.1 architectures. CICS MRO and IPIC use the APPC recovery protocols. Although the formats of syncpoint flows for APPC and LUTYPE6.1 are different, the concepts of syncpoint exchanges are similar.

In CICS, the flows involved in syncpoint exchanges are generated automatically in response to explicit or implicit SYNCPOINT commands issued by a transaction. However, a basic understanding of the flows that are involved can help you in the design of your application and give you an appreciation of the consequences of session or system failure during the syncpoint activity. For more information about these flows, see Syncpointing a distributed process.

Figures Figure 88 on page 299 through Figure 90 on page 300 show some examples of syncpoint flows. In the figures, the numbers in brackets, for example, (1), show the sequence of the actions in each flow.

A CICS task may contain one or more UOWs. A local UOW that initiates syncpoint activity—by, for example, issuing an EXEC CICS SYNCPOINT or an EXEC CICS RETURN command—is called an **initiator**. A local UOW that receives syncpoint requests from an initiator is called an **agent**. The simplest case is shown in Figure 88 on page 299. There is a single conversation between an initiator and an agent. At the start of the syncpoint activity, the initiator sends a **commit** request to the agent. The agent commits its changes and responds with **committed**. The initiator then commits its changes, and the unit of work is complete. However, the agent retains recovery information about the UOW until its partner tells it (by means of a forget flow) that the information can be discarded.

Between the commit flow and the committed flow, the initiator is indoubt, but the agent is not. The local UOW that is not indoubt is called the **coordinator**, because it coordinates the commitment of resources on both systems. The local UOW that is indoubt is called the **subordinate**, because it must obey the decision to commit or back out taken by its coordinator.

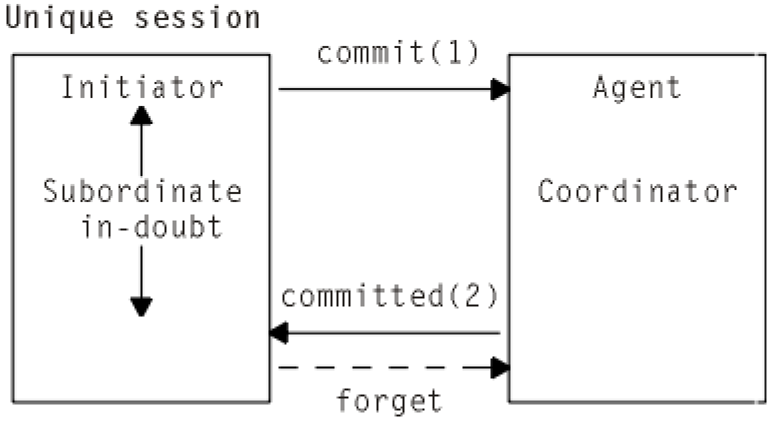


Figure 88. Syncpointing flows—unique session

Figure 89 on page 300 shows a more complex example. Here, the agent UOW (Agent1) has a conversation with a third local UOW (Agent2). Agent1 initiates syncpoint activity on this latter conversation before it responds to the initiator. Agent2 commits first, then Agent1, and finally the initiator. Note that, in Figure 89 on page 300, Agent1 is both the coordinator of the initiator and a subordinate of Agent2.

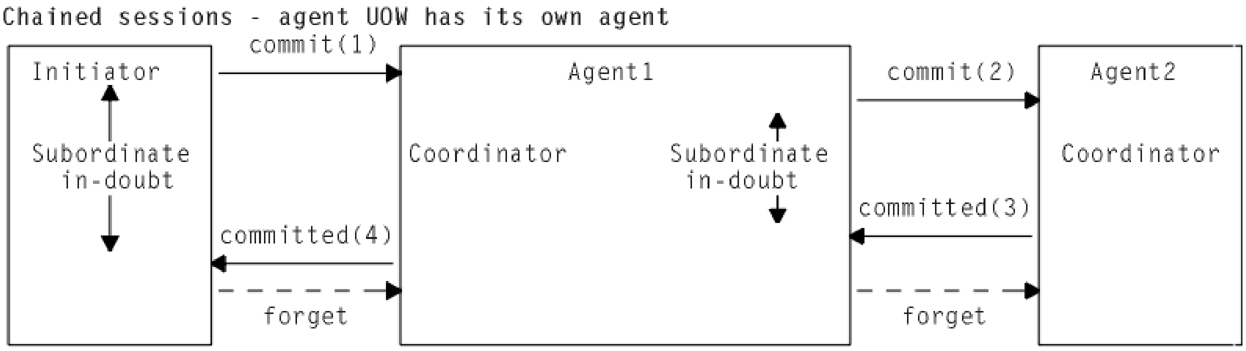


Figure 89. Syncpointing flows—chained sessions

Figure 90 on page 300 shows a more general case, in which the initiator UOW has more than one (directly-connected) agent. It must inform each of its agents that a syncpoint is being taken. It does this by sending a prepare to commit request to all of its agents except the last. The **last agent** is the agent that is not told to prepare to commit.

Note: CICS chooses the last agent dynamically, at the time the syncpoint is issued. CICS external interfaces do not provide a means of identifying the last agent.

Each agent that receives a prepar request responds with a commit request. When all such prepare requests have been sent and all the commit responses received, the initiator sends a commit request to its last agent. When this responds with a committed indication, the initiator then sends committed requests to all the other agents.

Note that, in Figure 90 on page 300, the Initiator is both the coordinator of Agent1 and a subordinate of Agent2. Agent2 is the last agent.

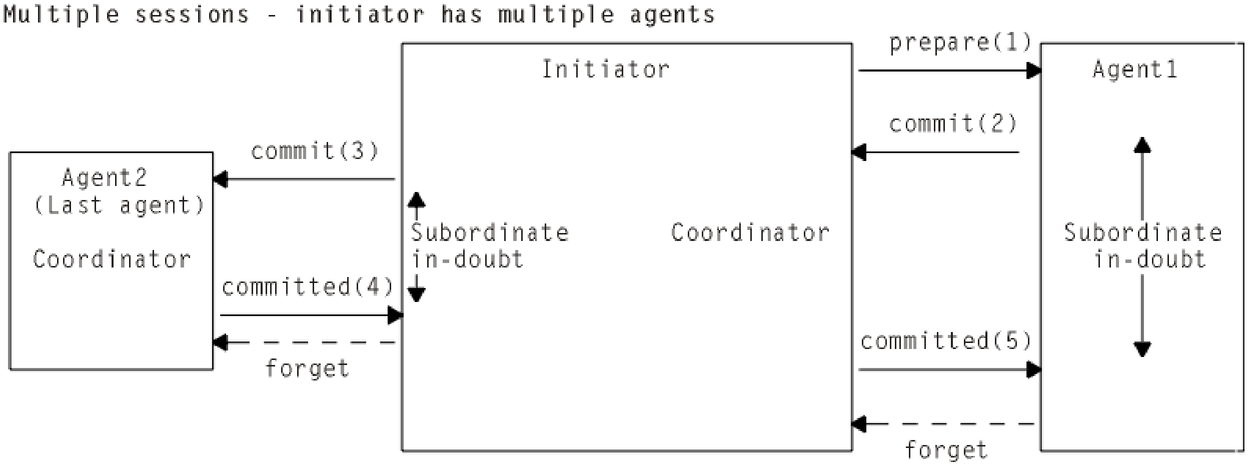


Figure 90. Syncpointing flows—multiple sessions

Recovery functions and interfaces

This section describes the functions and interfaces provided by CICS for recovery after a communication failure, or a CICS system failure.

Important:

Not all CICS releases provide the same level of support; this section describes MRO, IPIC, and ISC over SNA (APPC) parallel-session connections to other CICS Transaction Server for z/OS systems. Much of it applies also to other types of connection, but with some restrictions. For information about the restrictions for connections to non-CICS Transaction Server for z/OS systems, and for LU6.1 and APPC single-session connections, see “Connections that do not fully support shunting” on page 304.

This section also assumes that each CICS system is restarted correctly (that is, that AUTO is coded on the START system initialization parameter). If an initial start is performed there are implications for connected systems; these are described in "Initial and cold starts" on page 305.

Recovery functions

If CICS is indoubt about a unit of work because of a communication failure, it can either suspend commitment of updated resources, or decide to commit or back out local resources.

How you can influence which of these two actions CICS takes is described in "The indoubt attributes of the transaction definition" on page 301.

- Suspend commitment of updated resources until the systems are next in communication. The unit of work is *shunted*. When communication is restored, the decision to commit or back out is obtained from the coordinator system; the unit of work is *unshunted*, and the updates are committed or backed out on the local system in a process called *resynchronization*.
- Take a unilateral decision to commit or back out local resources. In this case, the decision may be inconsistent with other systems; at the restoration of communications, the decisions are compared and CICS warns of any inconsistency between neighboring systems (see "Messages that report CICS recovery actions" on page 291).

There is a trade-off between the two functions: the suspension of indoubt UOWs causes updated data to be locked against subsequent access; this disadvantage has to be weighed against the possibility of corruption of the consistency of data, which could result from taking unilateral decisions. When unilateral decisions are taken, there might be application-dependent processes, such as reconciliation jobs, that can restore consistency, but there is no general method that can be provided by CICS.

Recovery interfaces

This section summarizes the resource definition options, system programming commands, and CICS-supplied transactions that you can use to control and investigate units of work that fail during the indoubt period.

The indoubt attributes of the transaction definition

You can control the action that CICS takes after a communication failure during the indoubt period by specifying indoubt attributes when you define the transaction, using the WAIT, WAITTIME, and ACTION attributes of the TRANSACTION resource.

These options are honored when communication is lost with the coordinator and the UOW is in the indoubt period.

Use the following attributes of the TRANSACTION resource:

WAIT({YES|NO})
Specifies whether or not a unit of work is to wait, pending recovery from a failure that occurred after it had entered the indoubt period, before taking the action specified by ACTION.

YES
The UOW is to wait, pending recovery from the failure, to resolve its indoubt state and determine whether recoverable resources are to be backed out or committed. In other words, it is to be shunted.

NO
The UOW is not to wait. CICS takes immediately whatever action is specified on the ACTION attribute.

Note: The setting of the WAIT option can be overridden by other system settings—see TRANSACTION definition attributes.

WAITTIME({`00,00,00`|dd,hh,mm})
Specifies, if WAIT=YES, how long the transaction is to wait, before taking the action specified by ACTION.

You can use WAIT and WAITTIME to allow an opportunity for normal recovery and resynchronization to take place, while ensuring that a unit of work releases locks within a reasonable time.

ACTION({BACKOUT|COMMIT})

Specifies the action to be taken when communication with the coordinator of the unit of work is lost, and the UOW has entered the indoubt period.

BACKOUT

All changes made to recoverable resources are backed out, and the resources are returned to the state they were in before the start of the UOW.

COMMIT

All changes made to recoverable resources are committed and the UOW is marked as completed.

The action is dependent on the WAIT attribute. If WAIT specifies YES, ACTION has no effect unless the interval specified on the WAITTIME option expires before recovery from the failure.

Whether you specify BACKOUT or COMMIT is likely to depend on the kinds of changes that the transaction makes to resources in the remote system—see “Specifying indoubt attributes—an example” on page 302.

Specifying indoubt attributes—an example

This simple example is an illustration of specifying the indoubt attributes of a transaction.

Example:

A transaction is given a part number; it checks the entry in a local file to see whether the part is in stock, decrements the quantity in stock by updating the stock file, and sends a record to a remote transient data queue to initiate the dispatch of the part.

The update to the local file should take place only if the addition is made to the remote transient data (TD) queue, and the TD queue should only be updated if an update is made to the local file. The first step towards achieving this is to specify both the file and the TD queue as recoverable resources. This ensures synchronization of the changes to the resources (that is, both changes will either be backed out or committed) in all cases except for a session or system failure during the indoubt period of syncpoint processing.

To deal with a communications failure—for example, a failure of the remote system—during the indoubt period, specify on the local transaction definition, WAIT(YES), ACTION(BACKOUT), and a WAITTIME long enough to allow the remote system to be recycled. This enables resynchronization to take place automatically, if communication is restored within the specified time limit. During the WAITTIME period, until resynchronization takes place, the local UOW is shunted, and a lock is held on the stock-file record.

If communication is not restored within the time limit, changes made to the stock file on the local system are backed out. The addition to the TD queue on the remote system may or may not have been committed; this must be investigated after communication is restored.

INQUIRE commands

The CEMT and EXEC CICS interfaces provide a set of inquiry commands that you can use to investigate the execution of distributed units of work, and diagnose problems.

In the following list of commands, **`INQUIRE CONNECTION`** applies to MRO and ISC over SNA (APPC) connections. **`INQUIRE IPCONN`** applies to IPIC connections.

In CICS Explorer, the **ISC/MRO Connections** view and **IPIC Connections** view provide functional equivalents to the INQUIRE CONNECTION and INQUIRE IPCONN commands. See ISC/MRO Connections view in the CICS Explorer product documentation and IPIC Connections view in the CICS Explorer product documentation.

In summary, the commands are:

INQUIRE {CONNECTION | IPCONN} RECOVSTATUS

Use this command to find out whether any resynchronization work is outstanding between the local system and the connected system. The returned CVDA values are:

NORECOVDATA
Neither side has recovery information outstanding.

NOTAPPLIC
This is not an IPIC, APPC parallel-session, nor a CICS-to-CICS MRO connection, and does not support two-phase commit protocols.

NRS
CICS does not have recovery outstanding for the connection, but the partner may have.

RECOVDATA
There are indoubt units of work associated with the connection, or there are outstanding resyncs awaiting FORGET on the connection. Resynchronization takes place when the connection next becomes active, or when the UOW is unshunted.

INQUIRE {CONNECTION | IPCONN} PENDSTATUS
Use this command to discover whether there are any UOWs for which resynchronization is impossible because of an initial start by the connected system.

INQUIRE CONNECTION XLNSTATUS (APPC parallel-sessions only)
In CICS Explorer, the **ISC/MRO Connections** view provides a functional equivalent to this command. See ISC/MRO Connections view in the CICS Explorer product documentation

Use it to discover whether the link is currently able to support syncpoint (synclevel 2) work. See “The exchange lognames process” on page 307 for more information.

Note: XLNSTATUS is not applicable to IPCONNs.

INQUIRE UOW
Use this command to discover why a unit of work is waiting or shunted. If the reason is a connection failure (the WAITCAUSE option returns a CVDA value of CONNECTION), the SYSID and LINK options return the sysid and netname of the remote system that caused the UOW to wait or be shunted.

Note that INQUIRE UOW returns information about a *local* UOW: that is, for a distributed UOW it returns information only about the work required on the local system. You can assemble information about a distributed UOW by matching the network-wide identifier returned in the NETUOWID field against the identifiers of local UOWs on other systems. For an example of how to do this, see “Resolving a resynchronization failure” on page 294.

INQUIRE UOWLINK
This command allows you to inquire about the resynchronization needs of individual UOWs. Use it to discover information about connections involved in a distributed UOW.

For a local UOW, INQUIRE UOWLINK returns a list of tokens (*UOW-links*) representing connections to the systems that are involved in the distributed UOW. For each UOW-link, INQUIRE UOWLINK returns:

- The CONNECTION name
- The resynchronization status of the connection
- Whether the connection is to a coordinator or a subordinate system.

For examples of the use of these commands to diagnose problems with distributed units of work, see “Problem determination examples” on page 294.

SET {CONNECTION | IPCONN} command

In exceptional cases, you may need to override the indoubt action normally controlled by the transaction definition.

For example, a connected system may take longer than expected to restart. If the connected system is the coordinator of any UOWs, you can use the EXEC CICS or CEMT SET {CONNECTION | IPCONN} UOWACTION(FORCE|COMMIT|BACKOUT) command to force the UOWs to take a local, unilateral decision to commit or back out.

Note: **`SET CONNECTION`** applies to MRO and ISC over SNA (APPC) connections. **`SET IPCONN`** applies to IPIC (IP) connections.

The following commands are described in "The exchange lognames process" on page 307 and Administering connections between CICS systems:

- SET {CONNECTION | IPCONN} PENDSTATUS
- SET {CONNECTION | IPCONN} RECOVSTATUS.

Connections that do not fully support shunting

This section describes exceptions that apply, for example, to connections to back-level systems.

The information in previous sections assumes that you are using MRO, IPIC, or APPC parallel-session connections to other CICS Transaction Server for z/OS systems; that is, that your network consists solely of current systems that fully support shunting. Much of the preceding information applies equally to other types of connection.

LU6.1 connections

This section describes the ways in which LU6.1 connections differ from APPC parallel-session connections and MRO connections to CICS TS for z/OS systems.

Recovery functions and interfaces

Some recovery functions are not available to LU6.1 connections:

- Shunting is not always supported.
- Some recovery-related commands and options are not supported.
- Resynchronization takes place on a session-by-session basis.

Restriction on shunting support

There is no LU6.1 protocol by which one system can notify another system that a unit of work has been shunted. The only time when a UOW that includes an LU6.1 session can be shunted is when all the following are true:

- There is only one LU6.1 session in the local UOW.
- The LU6.1 session is the coordinator.
- The LU6.1 session has failed during the indoubt period.
- The LU6.1 session is to the last agent.

Under these conditions, the UOW can be shunted, because there is no need for the LU6.1 partner to be notified of the shunt.

Under other conditions, a UOW that fails in the indoubt period, and that involves an LU6.1 session, takes a unilateral decision. If WAIT(YES) is specified on the transaction definition, it has no effect—WAIT(NO) is forced.

Unsupported commands

The following commands are not supported on LU6.1 connections:

- INQUIRE CONNECTION PENDSTATUS
- INQUIRE CONNECTION RECOVSTATUS
- INQUIRE CONNECTION XLNSTATUS.

Lack of SYNCPOINT ROLLBACK support

There is no LU6.1 protocol by which one system can notify another that a UOW has been backed out, without terminating the conversation. An attempt to issue an EXEC CICS SYNCPOINT ROLLBACK command in a UOW that includes an LU6.1 session results in an ASP8 abend. This abend cannot be handled by the application program.

Any resources in the UOW are backed out, but the transaction is not able to continue.

Session-by-session resynchronization

Unlike APPC parallel-session connections and CICS TS for z/OS-CICS TS for z/OS MRO connections, LU6.1 sessions are resynchronized one by one, as they are bound. Therefore, any UOW that requires resynchronization is not resynchronized until the session that failed is reconnected.

Initial and cold starts

The LU6.1 connection definition contains sequence numbers used for recovery. If you perform an initial or cold start of CICS when there are LU6.1 connections on which recovery is outstanding, the sequence numbers are lost, and it becomes impossible for the partner systems to resynchronize their outstanding units of work.

Lognames are not used. Therefore, the XLNACTION attribute of the CONNECTION resource is meaningless for LU6.1 connections.

Managing connection definitions

Recovery information for a remote system is *not* stored independently from the connection definition for the system—the LU6.1 connection definition contains sequence numbers used for recovery. Therefore you should not modify or discard connections for which recovery information may be outstanding.

APPC connections to non-CICS TS for z/OS systems

Some non-CICS Transaction Server for z/OS systems that can be connected to by APPC links do not support shunting, and always take unilateral action if a session failure occurs during the indoubt period.

Inevitably, communication with a system that does not support shunting involves a risk of damage to data integrity through the taking of unilateral decisions. It is not possible for CICS to distinguish systems that do not support shunting from others that *do* support shunting. Therefore, it cannot preferentially select such a system to be the coordinator of a unit of work.

Note the following:

- When unshunting takes place, there may be some delay before the unshunting is communicated to the non-CICS TS for z/OS system.
- Sessions may be unbound by CICS or its partner system as a normal part of the shunting and resynchronization process.

APPC single-session connections

Normal syncpoint protocols cannot be used across a connection that is defined as SINGLESESS(YES).

If function shipping is used (inbound or outbound), CICS communicates the outcome of a unit of work. However, resynchronization cannot be performed in the case of session failure.

CICS issues a message to inform you of the shunting—but not the unshunting— of a unit of work.

If the connection to which a function-shipped request is made is defined as remote (that is, it is owned by a remote region), the connection to the remote region must be defined as a parallel-session link, if recovery protocols with the resource-owning system are to be enabled.

Initial and cold starts

This section describes functions to manage the exceptional conditions that can occur in a transaction-processing network when one system performs an initial or cold start.

Important:

- Except where otherwise stated, this section describes the effect of initial and cold starts on CICS Transaction Server for z/OS systems that are connected by MRO, IPIC, or ISC over SNA (APPC) parallel-session links. For information about the effects when other connections are used, see "Connections that do not fully support shunting" on page 304.

- In the rest of this section, the term *cold start* means a cold start in the CICS TS for z/OS meaning of the phrase (see cold start section). Where an *initial start* is intended, the term is used explicitly.

CICS Transaction Server for z/OS systems can be started without full recovery in two ways:

Initial start

An *initial start* can be performed in either of the following circumstances:

- INITIAL is specified on the **START** system initialization parameter.
- AUTO is specified on the **START** system initialization parameter, and the recovery manager utility program, DFHRMUTL, has been used to set the AUTOINIT autostart override in the global catalog.

On an initial start, all information about both local and remote resources is erased, and all resource definitions are reinstalled from the CSD or from CICS tables.

An initial start should be performed only in exceptional circumstances. Examples of times when an initial start is appropriate are:

- When bringing up a new CICS system for the first time
- After a serious software failure, when the global catalog or system log has been corrupted.

Cold start

A *cold start* can be performed in either of the following circumstances:

- COLD is specified on the **START** system initialization parameter.
- AUTO is specified on the **START** system initialization parameter, and the DFHRMUTL utility has been used to set the AUTOCOLD autostart override in the global catalog.

In CICS TS for z/OS, a cold start means that log information about *local* resources is erased, and resource definitions are reinstalled from the CSD or from CICS tables. However, resynchronization information relating to remote systems or to RMI-connected resource managers is preserved. The CICS log is scanned during startup, and information regarding unit of work obligations to remote systems, or to non-CICS resource managers (such as Db2) connected through the RMI, is preserved. (That is, any decisions about the outcome of local UOWs, needed to allow remote systems or RMI resource managers to resynchronize their resources, are preserved.)

For guidance information about the different ways in which CICS can be started, see Troubleshooting for recovery processing.

Deciding when a cold start is possible

At a cold start, information relating to intersystem recovery is read from the system log.

Connected systems act as if the local system restarted normally, and resynchronize any outstanding work. Note that updates to *local* resources that were not fully committed or backed out during the previous run of CICS are not recovered at a cold start, *even if the updates were part of a distributed unit of work*.

A cold start will not damage the integrity of data if **all** the following conditions are true:

1. *Either*

   - The local system has no local recoverable resources (a TOR, for example), *or*
   - The previous run of CICS was successfully quiesced (shutdown was normal rather than immediate) and no units of work were shunted.

   Note: On a normal shutdown, CICS issues messages to help you decide whether a cold start can be performed safely. If there are no shunted UOWs, CICS issues message DFHRM0204. If there *are* shunted UOWs, it issues message DFHRM0203—you should not perform a cold start.
2. Attached resource managers that use the RMI are subsequently reconnected to allow resynchronization.
3. Connections to remote systems required for resynchronization are subsequently acquired.

   A system which has been subjected to a cold start may or may not contain the same connection definitions that were in use at the previous shutdown. If autoinstalled connections are missing, the remote system may cause them to be re-created, in which case resynchronization takes place. If this does not happen—or the connection definitions are missing—some action must be taken. See Administering connections between CICS systems.

If you have defined the system to be part of a z/OS Communications Server generic resource group, its connections can be correctly reestablished, provided the affinity relationship maintained by z/OS Communications Server is still valid. However, the loss of autoinstalled definitions may make it difficult to end z/OS Communications Server affinities, if this is required. See APPC connections to and from z/OS Communications Server generic resources.

The DFHRMUTL utility returns information about the type of the last CICS shutdown which is of use in determining whether a cold restart is possible or not. For further details, see the *CICS Operations and Utilities Guide*.

The exchange lognames process

The protocols that control the communication of syncpointing commit and backout decisions depend on information in the system log.

Each time CICS systems connect they exchange tokens called **lognames**. Lognames are verified during resynchronization; an *exchange lognames failure* means that the recovery protocol has been corrupted. A failure can take two forms:

1. A **cold/warm log mismatch**. A cold/warm log mismatch is caused by the loss of log data at one partner when the other has resynchronization work outstanding.

   Note: The term *cold start* is used in z/OS Communications Server: SNA product documentation, and by other products that communicate with CICS TS for z/OS to describe the cause of a loss of log data.

   Cold start is also used in CICS TS for z/OS messages and interfaces to describe the action of a partner system that results in a loss of log data for CICS TS for z/OS.

   However, in CICS TS for z/OS, a loss of log data for connected systems is caused by an *initial* start (not by a cold start), or by a SET CONNECTION NORECOVDATA command.
2. A **lognames mismatch**. A lognames mismatch is caused by a corruption of logname data. This can occur due to:
   a. A system logic error
   b. An operational error—for example, a failure to perform an initial start when upgrading from a back-level CICS release to CICS Transaction Server for z/OS.

The exchange lognames process is defined by the APPC architecture. MRO and IPIC use a similar protocol to APPC, with the important difference that after the erasure of log information at a partner, they allow new work to begin whatever the condition of existing work. On APPC synclevel 2 sessions, no further work is possible until action has been taken to delete any outstanding resynchronization work.

After a partner system has been reconnected, you can use the INQUIRE CONNECTION PENDSTATUS command to check whether there is any outstanding resynchronization work that has been invalidated by the erasure of log information at the partner. A status of 'PENDING' indicates that there is. To check whether APPC connections are able to execute new synclevel 2 work, use the INQUIRE CONNECTION XLNSTATUS command. A status of 'XNOTDONE' indicates that the exchange lognames process has not completed successfully, probably because of a loss of log data.

When CICS detects that a partner system has lost log data, the possible actions it can take are:

1. None. If there is no resynchronization work outstanding on the local system, the loss of log data has no effect.
2. Keep outstanding resynchronization work (which may include UOWs which were indoubt when communication was lost) for investigation.
3. Delete outstanding resynchronization work; any indoubt UOWs are committed or backed out according to the ACTION option of the associated transaction definition, and any decisions remembered for the partner are forgotten.

When there is outstanding resynchronization work, you can control (for IPIC, MRO and APPC connections) which of actions 2 or 3 CICS takes:

- **Automatically**, using the XLNACTION option of the connection definition. To delete resynchronization work as soon as the loss of log data by the partner is detected, use XLNACTION(FORCE).
- **Manually**, using the SET UOW and SET CONNECTION PENDSTATUS(NOTPENDING) commands.

Considerations for APPC connections

The exchange lognames process affects only level 2 synchronization conversations. If it fails, synclevel 2 conversations are not allowed on the link until the failure is resolved by operator action. However, synclevel 0 and synclevel 1 traffic on the link is unaffected by the failure, and continues as normal.

Appendix A. CICS-supported conversions

The conversion groups for the supported Coded Character Set Identifiers (CCSIDs) are listed. CCSIDs are provided for code page conversion for use with the DFHCCNV conversion program.

For unsupported CCSIDs, you can create your own conversion tables for use with the DFHCCNV conversion program. See User-defined conversion tables.

For nonstandard conversions, you must supply your own conversion program. See "User/CICS conversion" on page 340.

In most cases, CICS Transaction Server for z/OS can convert character data between ASCII and EBCDIC if both CCSIDs are in the same conversion group. However, some conversions within a conversion group are not supported. For example, when new CCSIDs are defined to extend the character set, conversions between new equivalent ASCII and EBCDIC CCSIDs are supported, but conversions that mix old and new ASCII and EBCDIC CCSIDs might not be supported. An example of this situation is a character set that is extended to include the euro.

Table 34. Conversion groups

Group	Countries or regions
Arabic	
Baltic Rim	Latvia, Lithuania, Estonia
Cyrillic	Eastern Europe; Bulgaria, Russia, Yugoslavia
Devanagari (Hindi)	India
Farsi (Persian)	Iran
Greek	Greece
Hebrew	Israel
Japanese	Japan
Korean	Korea
Lao	Laos
Latin-1 Latin-9	USA, Western Europe, and many other countries
Latin-2	Eastern Europe; Albania, Czech Republic, Hungary, Poland, Romania, Slovakia, Yugoslavia, Former Yugoslavia
Latin-5	Turkey
Simplified Chinese	Peoples' Republic of China
Thai	Thailand
Traditional Chinese	Taiwan
Urdu	Pakistan
Vietnamese	Vietnam

The tables in the following sections list the CCSIDs supported for each group. For each CCSID, they show:

- The value to be specified for the CLINTCP or SRVERCP keyword.
- The code page identifier or identifiers (CPGIDs).

- An IANA-registered character set name for the code page, where a suitable name exists and CICS supports the use of this name on EXEC CICS commands. The CICS-supported name might be the primary name or a preferred alias. In some cases, more than one name or alias is supported.

Arabic

The Coded Character Set Identifiers (CCSIDs) for Arabic conversions are listed.

Data conversion does not change the direction of Arabic data.

Table 35. Arabic, Client CCSIDs

CLINTCP	CCSID	CPGID	IANA character set name	Comments
864	00864	00864	ibm864	PC data: Arabic
1089 8859-6	01089	01089	iso-8859-6 iso_8859-6	ISO 8859-6: Arabic
1256	01256	01256	windows-1256	MS Windows: Arabic
5352	05352	01256		MS Windows: Arabic, version 2 with euro
9448	09448	09448		MS Windows: Arabic, 2001
17248	17248	00864		PC Data: Arabic with euro

Table 36. Arabic, Server CCSIDs

SRVERCP	CCSID	CPGID	IANA character set name	Comments
420	00420	00420	ibm420	Host: Arabic
16804	16804	00420		Host: Arabic with euro

Related reference

Baltic Rim
The Coded Character Set Identifiers (CCSIDs) for Baltic Rim conversions are listed.

Cyrillic
The Coded Character Set Identifiers (CCSIDs) for Cyrillic conversions are listed.

Devanagari
The Coded Character Set Identifiers (CCSIDs) for Devanagari conversions are listed.

Farsi
The Coded Character Set Identifiers (CCSIDs) for Farsi conversions are listed.

Greek
The Coded Character Set Identifiers (CCSIDs) for Greek conversions are listed.

Hebrew
The Coded Character Set Identifiers (CCSIDs) for Hebrew conversions are listed.

Japanese
The Coded Character Set Identifiers (CCSIDs) for Japanese conversions are listed.

Korean
The Coded Character Set Identifiers (CCSIDs) for Korean conversions are listed.

Lao
The Coded Character Set Identifiers (CCSIDs) for Lao conversions are listed.

Latin-1 and Latin-9
The Coded Character Set Identifiers (CCSIDs) for Latin-1 and Latin-9 conversions are listed.

Latin-2

The Coded Character Set Identifiers (CCSIDs) for Latin-2 conversions are listed.

Latin-5
The Coded Character Set Identifiers (CCSIDs) for Latin-5 conversions are listed.

Simplified Chinese
The Coded Character Set Identifiers (CCSIDs) for Simplified Chinese conversions are listed.

Thai
The Coded Character Set Identifiers (CCSIDs) for Thai conversions are listed.

Traditional Chinese
The Coded Character Set Identifiers (CCSIDs) for Traditional Chinese conversions are listed.

Urdu
The Coded Character Set Identifiers (CCSIDs) for Urdu conversions are listed.

Vietnamese
The Coded Character Set Identifiers (CCSIDs) for Vietnamese conversions are listed.

Unicode data
CICS Transaction Server for z/OS provides limited support for Unicode-encoded character data. Workstations can share UCS-2 or UTF-8 encoded data with CICS Transaction Server for z/OS provided that no conversion is required.

Baltic Rim

The Coded Character Set Identifiers (CCSIDs) for Baltic Rim conversions are listed.

Table 37. Baltic Rim, Client CCSIDs

CLINTCP	CCSID	CPGID	IANA character set name	Comments
901	00901	00901		PC data: Latvia, Lithuania; with euro
902	00902	00902		PC data: Estonia with euro
921	00921	00921		PC data: Latvia, Lithuania
922	00922	00922		PC data: Estonia
1257	01257	01257	windows-1257	MS Windows: Baltic Rim
5353	05353	01257		MS Windows: Baltic Rim, version 2 with euro

Table 38. Baltic Rim, Server CCSIDs

SRVERCP	CCSID	CPGID	IANA character set name	Comments
1112	01112	01112		Host: Latvia, Lithuania
1122	01122	01122		Host: Estonia
1156	01156	01156		Host: Latvia, Lithuania; with euro
1157	01157	01157		Host: Estonia, with euro

Related reference

Arabic
The Coded Character Set Identifiers (CCSIDs) for Arabic conversions are listed.

Cyrillic
The Coded Character Set Identifiers (CCSIDs) for Cyrillic conversions are listed.

Devanagari

The Coded Character Set Identifiers (CCSIDs) for Devanagari conversions are listed.

Farsi
The Coded Character Set Identifiers (CCSIDs) for Farsi conversions are listed.

Greek
The Coded Character Set Identifiers (CCSIDs) for Greek conversions are listed.

Hebrew
The Coded Character Set Identifiers (CCSIDs) for Hebrew conversions are listed.

Japanese
The Coded Character Set Identifiers (CCSIDs) for Japanese conversions are listed.

Korean
The Coded Character Set Identifiers (CCSIDs) for Korean conversions are listed.

Lao
The Coded Character Set Identifiers (CCSIDs) for Lao conversions are listed.

Latin-1 and Latin-9
The Coded Character Set Identifiers (CCSIDs) for Latin-1 and Latin-9 conversions are listed.

Latin-2
The Coded Character Set Identifiers (CCSIDs) for Latin-2 conversions are listed.

Latin-5
The Coded Character Set Identifiers (CCSIDs) for Latin-5 conversions are listed.

Simplified Chinese
The Coded Character Set Identifiers (CCSIDs) for Simplified Chinese conversions are listed.

Thai
The Coded Character Set Identifiers (CCSIDs) for Thai conversions are listed.

Traditional Chinese
The Coded Character Set Identifiers (CCSIDs) for Traditional Chinese conversions are listed.

Urdu
The Coded Character Set Identifiers (CCSIDs) for Urdu conversions are listed.

Vietnamese
The Coded Character Set Identifiers (CCSIDs) for Vietnamese conversions are listed.

Unicode data
CICS Transaction Server for z/OS provides limited support for Unicode-encoded character data. Workstations can share UCS-2 or UTF-8 encoded data with CICS Transaction Server for z/OS provided that no conversion is required.

Cyrillic

The Coded Character Set Identifiers (CCSIDs) for Cyrillic conversions are listed.

Table 39. Cyrillic, Client CCSIDs

CLINTCP	CCSID	CPGID	IANA character set name	Comments
808	00808	00808		PC data: Cyrillic, Russia; with euro
848	00848	00848		PC data: Cyrillic, Ukraine; with euro
849	00849	00849		PC data: Cyrillic, Belarus; with euro
855	00855	00855	ibm855	PC data: Cyrillic
866	00866	00866	ibm866	PC data: Cyrillic, Russia
872	00872	00872		PC data: Cyrillic with euro

Table 39. Cyrillic, Client CCSIDs (continued)

CLINTCP	CCSID	CPGID	IANA character set name	Comments
915 8859-5	00915	00915	iso-8859-5 iso_8859-5	ISO 8859-5: Cyrillic
1124	01124	01124		8-bit: Cyrillic, Belarus
1125	01125	01125		PC Data: Cyrillic, Ukraine
1131	01131	01131		PC Data: Cyrillic, Belarus
1251	01251	01251	windows-1251	MS Windows: Cyrillic
5347	05347	01251		MS Windows: Cyrillic, version 2 with euro

Table 40. Cyrillic, Server CCSIDs

SRVERCP	CCSID	CPGID	IANA character set name	Comments
1025	01025	01025		Host: Cyrillic multilingual
1123	01123	01123		Host: Cyrillic Ukraine
1154	01154	01154		Host: Cyrillic multilingual; with euro
1158	01158	01158		Host: Cyrillic Ukraine; wtih euro

Related reference

Arabic
The Coded Character Set Identifiers (CCSIDs) for Arabic conversions are listed.

Baltic Rim
The Coded Character Set Identifiers (CCSIDs) for Baltic Rim conversions are listed.

Devanagari
The Coded Character Set Identifiers (CCSIDs) for Devanagari conversions are listed.

Farsi
The Coded Character Set Identifiers (CCSIDs) for Farsi conversions are listed.

Greek
The Coded Character Set Identifiers (CCSIDs) for Greek conversions are listed.

Hebrew
The Coded Character Set Identifiers (CCSIDs) for Hebrew conversions are listed.

Japanese
The Coded Character Set Identifiers (CCSIDs) for Japanese conversions are listed.

Korean
The Coded Character Set Identifiers (CCSIDs) for Korean conversions are listed.

Lao
The Coded Character Set Identifiers (CCSIDs) for Lao conversions are listed.

Latin-1 and Latin-9
The Coded Character Set Identifiers (CCSIDs) for Latin-1 and Latin-9 conversions are listed.

Latin-2
The Coded Character Set Identifiers (CCSIDs) for Latin-2 conversions are listed.

Latin-5
The Coded Character Set Identifiers (CCSIDs) for Latin-5 conversions are listed.

Simplified Chinese

The Coded Character Set Identifiers (CCSIDs) for Simplified Chinese conversions are listed.

Thai
The Coded Character Set Identifiers (CCSIDs) for Thai conversions are listed.

Traditional Chinese
The Coded Character Set Identifiers (CCSIDs) for Traditional Chinese conversions are listed.

Urdu
The Coded Character Set Identifiers (CCSIDs) for Urdu conversions are listed.

Vietnamese
The Coded Character Set Identifiers (CCSIDs) for Vietnamese conversions are listed.

Unicode data
CICS Transaction Server for z/OS provides limited support for Unicode-encoded character data. Workstations can share UCS-2 or UTF-8 encoded data with CICS Transaction Server for z/OS provided that no conversion is required.

Devanagari

The Coded Character Set Identifiers (CCSIDs) for Devanagari conversions are listed.

These Devanagari CCSIDs can also be used to encode the identical Devanagari character repertoire used by Marathi.

Table 41. Devanagari, Server CCSIDs

SRVERCP	CCSID	CPGID	IANA character set name	Comments
1137	01137	01137		Host: Devanagari

Table 42. Devanagari, Client CCSIDs

CLINTCP	CCSID	CPGID	IANA character set name	Comments
806	00806	00806		PC data: ISCII-91, Devanagari script code

Related reference

Arabic
The Coded Character Set Identifiers (CCSIDs) for Arabic conversions are listed.

Baltic Rim
The Coded Character Set Identifiers (CCSIDs) for Baltic Rim conversions are listed.

Cyrillic
The Coded Character Set Identifiers (CCSIDs) for Cyrillic conversions are listed.

Farsi
The Coded Character Set Identifiers (CCSIDs) for Farsi conversions are listed.

Greek
The Coded Character Set Identifiers (CCSIDs) for Greek conversions are listed.

Hebrew
The Coded Character Set Identifiers (CCSIDs) for Hebrew conversions are listed.

Japanese
The Coded Character Set Identifiers (CCSIDs) for Japanese conversions are listed.

Korean
The Coded Character Set Identifiers (CCSIDs) for Korean conversions are listed.

Lao

The Coded Character Set Identifiers (CCSIDs) for Lao conversions are listed.

Latin-1 and Latin-9
The Coded Character Set Identifiers (CCSIDs) for Latin-1 and Latin-9 conversions are listed.

Latin-2
The Coded Character Set Identifiers (CCSIDs) for Latin-2 conversions are listed.

Latin-5
The Coded Character Set Identifiers (CCSIDs) for Latin-5 conversions are listed.

Simplified Chinese
The Coded Character Set Identifiers (CCSIDs) for Simplified Chinese conversions are listed.

Thai
The Coded Character Set Identifiers (CCSIDs) for Thai conversions are listed.

Traditional Chinese
The Coded Character Set Identifiers (CCSIDs) for Traditional Chinese conversions are listed.

Urdu
The Coded Character Set Identifiers (CCSIDs) for Urdu conversions are listed.

Vietnamese
The Coded Character Set Identifiers (CCSIDs) for Vietnamese conversions are listed.

Unicode data
CICS Transaction Server for z/OS provides limited support for Unicode-encoded character data. Workstations can share UCS-2 or UTF-8 encoded data with CICS Transaction Server for z/OS provided that no conversion is required.

Farsi

The Coded Character Set Identifiers (CCSIDs) for Farsi conversions are listed.

Data conversion does not change the direction of Farsi data.

Table 43. Farsi, Client CCSIDs

CLINTCP	CCSID	CPGID	IANA character set name	Comments
1098	01098	01098		PC data: Farsi

Table 44. Farsi, Server CCSIDs

SRVERCP	CCSID	CPGID	IANA character set name	Comments
1097	01097	01097		Host: Farsi

Related reference

Arabic
The Coded Character Set Identifiers (CCSIDs) for Arabic conversions are listed.

Baltic Rim
The Coded Character Set Identifiers (CCSIDs) for Baltic Rim conversions are listed.

Cyrillic
The Coded Character Set Identifiers (CCSIDs) for Cyrillic conversions are listed.

Devanagari
The Coded Character Set Identifiers (CCSIDs) for Devanagari conversions are listed.

Greek
The Coded Character Set Identifiers (CCSIDs) for Greek conversions are listed.

Hebrew

The Coded Character Set Identifiers (CCSIDs) for Hebrew conversions are listed.

Japanese
The Coded Character Set Identifiers (CCSIDs) for Japanese conversions are listed.

Korean
The Coded Character Set Identifiers (CCSIDs) for Korean conversions are listed.

Lao
The Coded Character Set Identifiers (CCSIDs) for Lao conversions are listed.

Latin-1 and Latin-9
The Coded Character Set Identifiers (CCSIDs) for Latin-1 and Latin-9 conversions are listed.

Latin-2
The Coded Character Set Identifiers (CCSIDs) for Latin-2 conversions are listed.

Latin-5
The Coded Character Set Identifiers (CCSIDs) for Latin-5 conversions are listed.

Simplified Chinese
The Coded Character Set Identifiers (CCSIDs) for Simplified Chinese conversions are listed.

Thai
The Coded Character Set Identifiers (CCSIDs) for Thai conversions are listed.

Traditional Chinese
The Coded Character Set Identifiers (CCSIDs) for Traditional Chinese conversions are listed.

Urdu
The Coded Character Set Identifiers (CCSIDs) for Urdu conversions are listed.

Vietnamese
The Coded Character Set Identifiers (CCSIDs) for Vietnamese conversions are listed.

Unicode data
CICS Transaction Server for z/OS provides limited support for Unicode-encoded character data. Workstations can share UCS-2 or UTF-8 encoded data with CICS Transaction Server for z/OS provided that no conversion is required.

Greek

The Coded Character Set Identifiers (CCSIDs) for Greek conversions are listed.

Table 45. Greek, Client CCSIDs

CLINTCP	CCSID	CPGID	IANA character set name	Comments
813 8859-7	00813	00813	iso-8859-7 iso_8859-7	ISO 8859-7: Greece
869	00869	00869	ibm869	PC data: Greece
1253	01253	01253	windows-1253	MS Windows: Greece
4909	04909	00813		ISO 8859-7: Greece with euro
5349	05349	01253		MS Windows: Greece, version 2 with euro
9061	09061	00869		PC Data: Greece with euro

Table 46. Greek, Server CCSIDs

SRVERCP	CCSID	CPGID	IANA character set name	Comments
875	00875	00875		Host: Greece

Table 46. Greek, Server CCSIDs (continued)

SRVERCP	CCSID	CPGID	IANA character set name	Comments
4971	04971	00875		Host: Greece with euro

Related reference

Arabic
The Coded Character Set Identifiers (CCSIDs) for Arabic conversions are listed.

Baltic Rim
The Coded Character Set Identifiers (CCSIDs) for Baltic Rim conversions are listed.

Cyrillic
The Coded Character Set Identifiers (CCSIDs) for Cyrillic conversions are listed.

Devanagari
The Coded Character Set Identifiers (CCSIDs) for Devanagari conversions are listed.

Farsi
The Coded Character Set Identifiers (CCSIDs) for Farsi conversions are listed.

Hebrew
The Coded Character Set Identifiers (CCSIDs) for Hebrew conversions are listed.

Japanese
The Coded Character Set Identifiers (CCSIDs) for Japanese conversions are listed.

Korean
The Coded Character Set Identifiers (CCSIDs) for Korean conversions are listed.

Lao
The Coded Character Set Identifiers (CCSIDs) for Lao conversions are listed.

Latin-1 and Latin-9
The Coded Character Set Identifiers (CCSIDs) for Latin-1 and Latin-9 conversions are listed.

Latin-2
The Coded Character Set Identifiers (CCSIDs) for Latin-2 conversions are listed.

Latin-5
The Coded Character Set Identifiers (CCSIDs) for Latin-5 conversions are listed.

Simplified Chinese
The Coded Character Set Identifiers (CCSIDs) for Simplified Chinese conversions are listed.

Thai
The Coded Character Set Identifiers (CCSIDs) for Thai conversions are listed.

Traditional Chinese
The Coded Character Set Identifiers (CCSIDs) for Traditional Chinese conversions are listed.

Urdu
The Coded Character Set Identifiers (CCSIDs) for Urdu conversions are listed.

Vietnamese
The Coded Character Set Identifiers (CCSIDs) for Vietnamese conversions are listed.

Unicode data
CICS Transaction Server for z/OS provides limited support for Unicode-encoded character data. Workstations can share UCS-2 or UTF-8 encoded data with CICS Transaction Server for z/OS provided that no conversion is required.

Hebrew

The Coded Character Set Identifiers (CCSIDs) for Hebrew conversions are listed.

Data conversion does not change the direction of Hebrew data.

Table 47. Hebrew, Client CCSIDs

CLINTCP	CCSID	CPGID	IANA character set name	Comments
856	00856	00856		PC data: Hebrew
862	00862	00862	ibm862	PC data: Hebrew (migration)
867	00867	00867		PC Data: Hebrew with euro
916 8859-8	00916	00916	iso-8859-8 iso_8859-8	ISO 8859-8: Hebrew
1255	01255	01255	windows-1255	MS Windows: Hebrew
5351	05351	01255		MS Windows: Hebrew, version 2 with euro
9447	09447	01255		MS Windows: Hebrew, version 2 with euro and new sheqel

Table 48. Hebrew, Server CCSIDs

SRVERCP	CCSID	CPGID	IANA character set name	Comments
424	00424	00424	ibm424	Host: Hebrew
803	00803	00803		Host: Hebrew (Character Set A)
4899	04899	00803		Host: Hebrew (Character Set A) with euro
12712	12712	00424		Host: Hebrew with euro and new sheqel

Related reference

Arabic
The Coded Character Set Identifiers (CCSIDs) for Arabic conversions are listed.

Baltic Rim
The Coded Character Set Identifiers (CCSIDs) for Baltic Rim conversions are listed.

Cyrillic
The Coded Character Set Identifiers (CCSIDs) for Cyrillic conversions are listed.

Devanagari
The Coded Character Set Identifiers (CCSIDs) for Devanagari conversions are listed.

Farsi
The Coded Character Set Identifiers (CCSIDs) for Farsi conversions are listed.

Greek
The Coded Character Set Identifiers (CCSIDs) for Greek conversions are listed.

Japanese
The Coded Character Set Identifiers (CCSIDs) for Japanese conversions are listed.

Korean
The Coded Character Set Identifiers (CCSIDs) for Korean conversions are listed.

Lao
The Coded Character Set Identifiers (CCSIDs) for Lao conversions are listed.

Latin-1 and Latin-9
The Coded Character Set Identifiers (CCSIDs) for Latin-1 and Latin-9 conversions are listed.

Latin-2

The Coded Character Set Identifiers (CCSIDs) for Latin-2 conversions are listed.

Latin-5
The Coded Character Set Identifiers (CCSIDs) for Latin-5 conversions are listed.

Simplified Chinese
The Coded Character Set Identifiers (CCSIDs) for Simplified Chinese conversions are listed.

Thai
The Coded Character Set Identifiers (CCSIDs) for Thai conversions are listed.

Traditional Chinese
The Coded Character Set Identifiers (CCSIDs) for Traditional Chinese conversions are listed.

Urdu
The Coded Character Set Identifiers (CCSIDs) for Urdu conversions are listed.

Vietnamese
The Coded Character Set Identifiers (CCSIDs) for Vietnamese conversions are listed.

Unicode data
CICS Transaction Server for z/OS provides limited support for Unicode-encoded character data. Workstations can share UCS-2 or UTF-8 encoded data with CICS Transaction Server for z/OS provided that no conversion is required.

Japanese

The Coded Character Set Identifiers (CCSIDs) for Japanese conversions are listed.

Table 49. Japanese, Client CCSIDs

CLINTCP	CCSID	CPGID	IANA character set name	Comments
932	00932	1. 00897 2. 00301		1. PC data: SBCS 2. PC data: DBCS including 1880 user-defined characters
942	00942	1. 01041 2. 00301		1. PC data: Extended SBCS 2. PC data: DBCS including 1880 user-defined characters
943	00943	1. 00897 2. 00941	shift-jis x-sjis	1. PC data: SBCS 2. PC data: DBCS for Open environment including 1880 IBM user-defined characters
954 EUCJP	00954	1. 00895 2. 00952 3. 00896 4. 00953	euc-jp	1. G0: JIS X201 Roman 2. G1: JIS X208-1990 3. G1: JIS X201 Katakana 4. G1: JIS X212
5050	05050	1. 00895 2. 00952 3. 00896 4. 00953		1. G0: JIS X201 Roman 2. G1: JIS X208-1990 3. G1: JIS X201 Katakana 4. G1: JIS X212

Table 50. Japanese, Server CCSIDs

SRVERCP	CCSID	CPGID	IANA character set name	Comments
930	00930	1. 00290 2. 00300 3. 00290 4. 00300		1. Katakana Host: extended SBCS 2. Kanji Host: DBCS including 4370 user-defined characters 3. Katakana Host: extended SBCS 4. Kanji Host: DBCS including 1880 user-defined characters
931	00931	1. 00037 2. 00300		1. Latin Host: SBCS 2. Kanji Host: DBCS including 4370 user-defined characters
939	00939	1. 01027 2. 00300 3. 01027 4. 00300		1. Latin Host: extended SBCS 2. Kanji Host: DBCS including 4370 user-defined characters 3. Latin Host: extended SBCS 4. Kanji Host: DBCS including 1880 user-defined characters
1390	01390	1. 00290 2. 00300		1. Katakana Host: extended SBCS; with euro 2. Kanji Host: DBCS including 6205 user-defined characters
1399	01399	1. 01027 2. 00300		1. Latin Host: extended SBCS; with euro 2. Kanji Host: DBCS including 4370 user-defined characters; with euro

Related reference

Arabic
The Coded Character Set Identifiers (CCSIDs) for Arabic conversions are listed.

Baltic Rim
The Coded Character Set Identifiers (CCSIDs) for Baltic Rim conversions are listed.

Cyrillic
The Coded Character Set Identifiers (CCSIDs) for Cyrillic conversions are listed.

Devanagari
The Coded Character Set Identifiers (CCSIDs) for Devanagari conversions are listed.

Farsi
The Coded Character Set Identifiers (CCSIDs) for Farsi conversions are listed.

Greek
The Coded Character Set Identifiers (CCSIDs) for Greek conversions are listed.

Hebrew
The Coded Character Set Identifiers (CCSIDs) for Hebrew conversions are listed.

Korean
The Coded Character Set Identifiers (CCSIDs) for Korean conversions are listed.

Lao

The Coded Character Set Identifiers (CCSIDs) for Lao conversions are listed.

Latin-1 and Latin-9
The Coded Character Set Identifiers (CCSIDs) for Latin-1 and Latin-9 conversions are listed.

Latin-2
The Coded Character Set Identifiers (CCSIDs) for Latin-2 conversions are listed.

Latin-5
The Coded Character Set Identifiers (CCSIDs) for Latin-5 conversions are listed.

Simplified Chinese
The Coded Character Set Identifiers (CCSIDs) for Simplified Chinese conversions are listed.

Thai
The Coded Character Set Identifiers (CCSIDs) for Thai conversions are listed.

Traditional Chinese
The Coded Character Set Identifiers (CCSIDs) for Traditional Chinese conversions are listed.

Urdu
The Coded Character Set Identifiers (CCSIDs) for Urdu conversions are listed.

Vietnamese
The Coded Character Set Identifiers (CCSIDs) for Vietnamese conversions are listed.

Unicode data
CICS Transaction Server for z/OS provides limited support for Unicode-encoded character data. Workstations can share UCS-2 or UTF-8 encoded data with CICS Transaction Server for z/OS provided that no conversion is required.

Korean

The Coded Character Set Identifiers (CCSIDs) for Korean conversions are listed.

Table 51. Korean, Client CCSIDs

CLINTCP	CCSID	CPGID	IANA character set name	Comments
934	00934	1. 00891 2. 00926		1. PC data: SBCS 2. PC data: DBCS including 1880 user-defined characters
944	00944	1. 01040 2. 00926		1. PC data: Extended SBCS 2. PC data: DBCS including 1880 user-defined characters
949	00949	1. 01088 2. 00951		1. IBM KS Code - PC data: SBCS 2. IBM KS code - PC data: DBCS including 1880 user-defined characters
970 EUCKR	00970	1. 00367 2. 00971	euc-kr	1. G0: ASCII 2. G1: KSC X5601-1989 including 1880 user-defined characters
1363	01363	1. 01126 2. 01362		1. PC data: MS Windows Korean SBCS 2. PC data: MS Windows Koran DBCS including 11172 full Hangul

Table 52. Korean, Server CCSIDs

SRVERCP	CCSID	CPGID	IANA character set name	Comments
933	00933	1. 00833 2. 00834		1. Host: Extended SBCS 2. Host: DBCS including 1880 user-defined characters and 11172 full Hangul characters
1364	01364	1. 00833 2. 00834		1. Host: Extended SBCS 2. Host: DBCS including 1880 user-defined characters and 11172 full Hangul characters

Related reference

Arabic
The Coded Character Set Identifiers (CCSIDs) for Arabic conversions are listed.

Baltic Rim
The Coded Character Set Identifiers (CCSIDs) for Baltic Rim conversions are listed.

Cyrillic
The Coded Character Set Identifiers (CCSIDs) for Cyrillic conversions are listed.

Devanagari
The Coded Character Set Identifiers (CCSIDs) for Devanagari conversions are listed.

Farsi
The Coded Character Set Identifiers (CCSIDs) for Farsi conversions are listed.

Greek
The Coded Character Set Identifiers (CCSIDs) for Greek conversions are listed.

Hebrew
The Coded Character Set Identifiers (CCSIDs) for Hebrew conversions are listed.

Japanese
The Coded Character Set Identifiers (CCSIDs) for Japanese conversions are listed.

Lao
The Coded Character Set Identifiers (CCSIDs) for Lao conversions are listed.

Latin-1 and Latin-9
The Coded Character Set Identifiers (CCSIDs) for Latin-1 and Latin-9 conversions are listed.

Latin-2
The Coded Character Set Identifiers (CCSIDs) for Latin-2 conversions are listed.

Latin-5
The Coded Character Set Identifiers (CCSIDs) for Latin-5 conversions are listed.

Simplified Chinese
The Coded Character Set Identifiers (CCSIDs) for Simplified Chinese conversions are listed.

Thai
The Coded Character Set Identifiers (CCSIDs) for Thai conversions are listed.

Traditional Chinese
The Coded Character Set Identifiers (CCSIDs) for Traditional Chinese conversions are listed.

Urdu
The Coded Character Set Identifiers (CCSIDs) for Urdu conversions are listed.

Vietnamese

The Coded Character Set Identifiers (CCSIDs) for Vietnamese conversions are listed.

Unicode data
CICS Transaction Server for z/OS provides limited support for Unicode-encoded character data. Workstations can share UCS-2 or UTF-8 encoded data with CICS Transaction Server for z/OS provided that no conversion is required.

Lao

The Coded Character Set Identifiers (CCSIDs) for Lao conversions are listed.

Table 53. Lao, Client CCSIDs

CLINTCP	CCSID	CPGID	IANA character set name	Comments
1133	01133	01133		ISO-8: Lao

Table 54. Lao, Server CCSIDs

SRVERCP	CCSID	CPGID	IANA character set name	Comments
1132	01132	01132		Host: Lao

Related reference

Arabic
The Coded Character Set Identifiers (CCSIDs) for Arabic conversions are listed.

Baltic Rim
The Coded Character Set Identifiers (CCSIDs) for Baltic Rim conversions are listed.

Cyrillic
The Coded Character Set Identifiers (CCSIDs) for Cyrillic conversions are listed.

Devanagari
The Coded Character Set Identifiers (CCSIDs) for Devanagari conversions are listed.

Farsi
The Coded Character Set Identifiers (CCSIDs) for Farsi conversions are listed.

Greek
The Coded Character Set Identifiers (CCSIDs) for Greek conversions are listed.

Hebrew
The Coded Character Set Identifiers (CCSIDs) for Hebrew conversions are listed.

Japanese
The Coded Character Set Identifiers (CCSIDs) for Japanese conversions are listed.

Korean
The Coded Character Set Identifiers (CCSIDs) for Korean conversions are listed.

Latin-1 and Latin-9
The Coded Character Set Identifiers (CCSIDs) for Latin-1 and Latin-9 conversions are listed.

Latin-2
The Coded Character Set Identifiers (CCSIDs) for Latin-2 conversions are listed.

Latin-5
The Coded Character Set Identifiers (CCSIDs) for Latin-5 conversions are listed.

Simplified Chinese
The Coded Character Set Identifiers (CCSIDs) for Simplified Chinese conversions are listed.

Thai
The Coded Character Set Identifiers (CCSIDs) for Thai conversions are listed.

Traditional Chinese

The Coded Character Set Identifiers (CCSIDs) for Traditional Chinese conversions are listed.

Urdu
The Coded Character Set Identifiers (CCSIDs) for Urdu conversions are listed.

Vietnamese
The Coded Character Set Identifiers (CCSIDs) for Vietnamese conversions are listed.

Unicode data
CICS Transaction Server for z/OS provides limited support for Unicode-encoded character data. Workstations can share UCS-2 or UTF-8 encoded data with CICS Transaction Server for z/OS provided that no conversion is required.

Latin-1 and Latin-9

The Coded Character Set Identifiers (CCSIDs) for Latin-1 and Latin-9 conversions are listed.

Note: In this group, conversions are supported between non euro-supported CCSIDs and euro-supported CCSIDs. However, use these conversions with care for the following reasons:

- The international currency symbol in each non euro-supported EBCDIC CCSID (for example, 00500) has been replaced by the euro symbol in the equivalent euro-supported EBCDIC CCSID (for example, 01148).
- The dotless *i* in non euro-supported ASCII CCSID 00850 has been replaced by the euro symbol in the equivalent euro-supported ASCII CCSID 00858.

Table 55. Latin-1, Client CCSIDs

CLINTCP	CCSID	CPGID	IANA character set name	Comments
437	00437	00437	ibm437	PC data: PC Base; USA, many other countries
819 8859-1	00819	00819	iso-8859-1 iso_8859-1	ISO 8859-1: Latin-1 countries
850	00850	00850	ibm850	PC data: Latin-1 countries
858	00858	00858	ibm00858	PC data: Latin-1 countries; with euro
923	00923	00923	iso-8859-15 iso_8859-15	ISO 8859-15: Latin-9
924	00924	00924	ibm00924	ISO 8859-15: Latin-9
1047	01047	01047		Host: Latin-1
1252	01252	01252	windows-1252	MS Windows: Latin-1 countries
5348	05348	01252		MS Windows: Latin-1 countries, version 2 with euro

Table 56. Latin-1 and Latin-9, Server CCSIDs

SRVERCP	CCSID	CPGID	IANA character set name	Comments
037	00037	00037	ibm037	Host: USA, Canada (ESA), Netherlands, Portugal, Brazil, Australia, New Zealand
273	00273	00273	ibm273	Host: Austria, Germany
277	00277	00277	ibm277	Host: Denmark, Norway
278	00278	00278	ibm278	Host: Finland, Sweden

Table 56. Latin-1 and Latin-9, Server CCSIDs (continued)				
SRVERCP	**CCSID**	**CPGID**	**IANA character set name**	**Comments**
280	00280	00280	ibm280	Host: Italy
284	00284	00284	ibm284	Host: Spain, Latin America (Spanish)
285	00285	00285	ibm285	Host: United Kingdom
297	00297	00297	ibm297	Host: France
500	00500	00500	ibm500	Host: Belgium, Canada (AS/400), Switzerland, International Latin-1
871	00871	00871	ibm871	Host: Iceland
924	00924	00924	ibm00924	Host: Latin-9
1047	01047	01047		Host: Latin-1
1140	01140	01140	ibm01140	Host: USA, Canada (ESA), Netherlands, Portugal, Brazil, Australia, New Zealand; with euro
1141	01141	01141	ibm01141	Host: Austria, Germany; with euro
1142	01142	01142	ibm01142	Host: Denmark, Norway; with euro
1143	01143	01143	ibm01143	Host: Finland, Sweden; with euro
1144	01144	01144	ibm01144	Host: Italy; with euro
1145	01145	01145	ibm01145	Host: Spain, Latin America (Spanish); with euro
1146	01146	01146	ibm01146	Host: United Kingdom; with euro
1147	01147	01147	ibm01147	Host: France; with euro
1148	01148	01148	ibm01148	Host: Belgium, Canada (AS/400), Switzerland, International Latin-1; with euro
1149	01149	01149	ibm01149	Host: Iceland; with euro

Related reference

Arabic
The Coded Character Set Identifiers (CCSIDs) for Arabic conversions are listed.

Baltic Rim
The Coded Character Set Identifiers (CCSIDs) for Baltic Rim conversions are listed.

Cyrillic
The Coded Character Set Identifiers (CCSIDs) for Cyrillic conversions are listed.

Devanagari
The Coded Character Set Identifiers (CCSIDs) for Devanagari conversions are listed.

Farsi
The Coded Character Set Identifiers (CCSIDs) for Farsi conversions are listed.

Greek
The Coded Character Set Identifiers (CCSIDs) for Greek conversions are listed.

Hebrew
The Coded Character Set Identifiers (CCSIDs) for Hebrew conversions are listed.

Japanese

The Coded Character Set Identifiers (CCSIDs) for Japanese conversions are listed.

Korean
The Coded Character Set Identifiers (CCSIDs) for Korean conversions are listed.

Lao
The Coded Character Set Identifiers (CCSIDs) for Lao conversions are listed.

Latin-2
The Coded Character Set Identifiers (CCSIDs) for Latin-2 conversions are listed.

Latin-5
The Coded Character Set Identifiers (CCSIDs) for Latin-5 conversions are listed.

Simplified Chinese
The Coded Character Set Identifiers (CCSIDs) for Simplified Chinese conversions are listed.

Thai
The Coded Character Set Identifiers (CCSIDs) for Thai conversions are listed.

Traditional Chinese
The Coded Character Set Identifiers (CCSIDs) for Traditional Chinese conversions are listed.

Urdu
The Coded Character Set Identifiers (CCSIDs) for Urdu conversions are listed.

Vietnamese
The Coded Character Set Identifiers (CCSIDs) for Vietnamese conversions are listed.

Unicode data
CICS Transaction Server for z/OS provides limited support for Unicode-encoded character data. Workstations can share UCS-2 or UTF-8 encoded data with CICS Transaction Server for z/OS provided that no conversion is required.

Latin-2

The Coded Character Set Identifiers (CCSIDs) for Latin-2 conversions are listed.

Conversions are supported for some combinations of Latin-2 ASCII CCSIDs and Latin-1 EBCDIC CCSIDs.

Table 57. Latin-2, Client CCSIDs

CLINTCP	CCSID	CPGID	IANA character set name	Comments
852	00852	00852	ibm852	PC data: Latin-2 multilingual
912 8859-2	00912	00912	iso-8859-2 iso_8859-2	ISO 8859-2: Latin-2 multilingual
1250	01250	01250	windows-1250	MS Windows: Latin-2
5346	05346	01250		MS Windows: Latin-2, version 2 with euro
9044	09044	00852		PC data: Latin-2 multilingual with euro

Table 58. Latin-2, Server CCSIDs

SRVERCP	CCSID	CPGID	IANA character set name	Comments
500	00500	00500	ibm500	Host: International Latin-1
870	00870	00870	ibm870	Host: Latin-2 multilingual
924	00924	00924	ibm00924	Host: Latin-9

Table 58. Latin-2, Server CCSIDs (continued)

SRVERCP	CCSID	CPGID	IANA character set name	Comments
1140	01140	01140	ibm01140	Host: USA, Canada (ESA), Netherlands, Portugal, Brazil, Australia, New Zealand; with euro
1141	01141	01141	ibm01141	Host: Austria, Germany; with euro
1142	01142	01142	ibm01142	Host: Denmark, Norway; with euro
1143	01143	01143	ibm01143	Host: Finland, Sweden; with euro
1144	01144	01144	ibm01144	Host: Italy; with euro
1145	01145	01145	ibm01145	Host: Spain, Latin America (Spanish); with euro
1146	01146	01146	ibm01146	Host: United Kingdom; with euro
1147	01147	01147	ibm01147	Host: France; with euro
1148	01148	01148	ibm01148	Host: International Latin-1 with euro
1149	01149	01149	ibm01149	Host: Iceland; with euro
1153	01153	01153		Host: Latin-2 multilingual with euro

Related reference

Arabic
The Coded Character Set Identifiers (CCSIDs) for Arabic conversions are listed.

Baltic Rim
The Coded Character Set Identifiers (CCSIDs) for Baltic Rim conversions are listed.

Cyrillic
The Coded Character Set Identifiers (CCSIDs) for Cyrillic conversions are listed.

Devanagari
The Coded Character Set Identifiers (CCSIDs) for Devanagari conversions are listed.

Farsi
The Coded Character Set Identifiers (CCSIDs) for Farsi conversions are listed.

Greek
The Coded Character Set Identifiers (CCSIDs) for Greek conversions are listed.

Hebrew
The Coded Character Set Identifiers (CCSIDs) for Hebrew conversions are listed.

Japanese
The Coded Character Set Identifiers (CCSIDs) for Japanese conversions are listed.

Korean
The Coded Character Set Identifiers (CCSIDs) for Korean conversions are listed.

Lao
The Coded Character Set Identifiers (CCSIDs) for Lao conversions are listed.

Latin-1 and Latin-9
The Coded Character Set Identifiers (CCSIDs) for Latin-1 and Latin-9 conversions are listed.

Latin-5
The Coded Character Set Identifiers (CCSIDs) for Latin-5 conversions are listed.

Simplified Chinese
The Coded Character Set Identifiers (CCSIDs) for Simplified Chinese conversions are listed.

Thai

The Coded Character Set Identifiers (CCSIDs) for Thai conversions are listed.

Traditional Chinese
The Coded Character Set Identifiers (CCSIDs) for Traditional Chinese conversions are listed.

Urdu
The Coded Character Set Identifiers (CCSIDs) for Urdu conversions are listed.

Vietnamese
The Coded Character Set Identifiers (CCSIDs) for Vietnamese conversions are listed.

Unicode data
CICS Transaction Server for z/OS provides limited support for Unicode-encoded character data. Workstations can share UCS-2 or UTF-8 encoded data with CICS Transaction Server for z/OS provided that no conversion is required.

Latin-5

The Coded Character Set Identifiers (CCSIDs) for Latin-5 conversions are listed.

Table 59. Latin-5, Client CCSIDs

CLINTCP	CCSID	CPGID	IANA character set name	Comments
857	00857	00857	ibm857	PC data: Latin-5 (Turkey)
920 8859-9	00920	00920	iso-8859-9 iso_8859-9	ISO 8859-9: Latin-5 (ECMA-128, Turkey TS-5881)
1254	01254	01254	windows-1254	MS Windows: Turkey
5350	05350	01254		MS Windows: Turkey, version 2 with euro
9049	09049	00857		PC data: Latin-5 (Turkey) with euro

Table 60. Latin-5, Server CCSIDs

SRVERCP	CCSID	CPGID	IANA character set name	Comments
1026	01026	01026	ibm1026	Host: Latin-5 (Turkey)
1155	01155	01155		Host: Latin-5 (Turkey) with euro

Related reference

Arabic
The Coded Character Set Identifiers (CCSIDs) for Arabic conversions are listed.

Baltic Rim
The Coded Character Set Identifiers (CCSIDs) for Baltic Rim conversions are listed.

Cyrillic
The Coded Character Set Identifiers (CCSIDs) for Cyrillic conversions are listed.

Devanagari
The Coded Character Set Identifiers (CCSIDs) for Devanagari conversions are listed.

Farsi
The Coded Character Set Identifiers (CCSIDs) for Farsi conversions are listed.

Greek
The Coded Character Set Identifiers (CCSIDs) for Greek conversions are listed.

Hebrew

The Coded Character Set Identifiers (CCSIDs) for Hebrew conversions are listed.

Japanese
The Coded Character Set Identifiers (CCSIDs) for Japanese conversions are listed.

Korean
The Coded Character Set Identifiers (CCSIDs) for Korean conversions are listed.

Lao
The Coded Character Set Identifiers (CCSIDs) for Lao conversions are listed.

Latin-1 and Latin-9
The Coded Character Set Identifiers (CCSIDs) for Latin-1 and Latin-9 conversions are listed.

Latin-2
The Coded Character Set Identifiers (CCSIDs) for Latin-2 conversions are listed.

Simplified Chinese
The Coded Character Set Identifiers (CCSIDs) for Simplified Chinese conversions are listed.

Thai
The Coded Character Set Identifiers (CCSIDs) for Thai conversions are listed.

Traditional Chinese
The Coded Character Set Identifiers (CCSIDs) for Traditional Chinese conversions are listed.

Urdu
The Coded Character Set Identifiers (CCSIDs) for Urdu conversions are listed.

Vietnamese
The Coded Character Set Identifiers (CCSIDs) for Vietnamese conversions are listed.

Unicode data
CICS Transaction Server for z/OS provides limited support for Unicode-encoded character data. Workstations can share UCS-2 or UTF-8 encoded data with CICS Transaction Server for z/OS provided that no conversion is required.

Simplified Chinese

The Coded Character Set Identifiers (CCSIDs) for Simplified Chinese conversions are listed.

Table 61. Simplified Chinese, Client CCSIDs

CLINTCP	CCSID	CPGID	IANA character set name	Comments
946	00946	1. 01042 2. 00928		1. PC data: Extended SBCS 2. PC data: DBCS including 1880 user-defined characters
1381	01381	1. 01115 2. 01380	gb2312	1. PC data: Extended SBCS (IBM GB) 2. PC data: DBCS (IBM GB) including 31 IBM-selected, 1880 user-defined characters
1383 EUCCN	01383	1. 00367 2. 01382		1. G0: ASCII 2. G1: GB 2312-80 set
1386	01386	1. 01114 2. 01385		1. PC data: S-Chinese GBK and T-Chinese IBM BIG-5 2. PC data: S-Chinese GBK

Table 61. Simplified Chinese, Client CCSIDs (continued)

CLINTCP	CCSID	CPGID	IANA character set name	Comments
5488	05488	1. 01252 2. 01385 3. 01391	gb18030	1. GB18030, 1-byte data 2. GB18030, 2-byte data 3. GB18030, 4-byte data

Table 62. Simplified Chinese, Server CCSIDs

SRVERCP	CCSID	CPGID	IANA character set name	Comments
935	00935	1. 00836 2. 00837		1. Host: Extended SBCS 2. Host: DBCS including 1880 user-defined characters
1388	01388	1. 00836 2. 00837		1. Host: Extended SBCS 2. Host: DBCS including 1880 user-defined characters
9127	09127	1. 00836 2. 00837		1. Host: Extended SBCS 2. Host: DBCS including 1880 user-defined characters

Related reference

Arabic
The Coded Character Set Identifiers (CCSIDs) for Arabic conversions are listed.

Baltic Rim
The Coded Character Set Identifiers (CCSIDs) for Baltic Rim conversions are listed.

Cyrillic
The Coded Character Set Identifiers (CCSIDs) for Cyrillic conversions are listed.

Devanagari
The Coded Character Set Identifiers (CCSIDs) for Devanagari conversions are listed.

Farsi
The Coded Character Set Identifiers (CCSIDs) for Farsi conversions are listed.

Greek
The Coded Character Set Identifiers (CCSIDs) for Greek conversions are listed.

Hebrew
The Coded Character Set Identifiers (CCSIDs) for Hebrew conversions are listed.

Japanese
The Coded Character Set Identifiers (CCSIDs) for Japanese conversions are listed.

Korean
The Coded Character Set Identifiers (CCSIDs) for Korean conversions are listed.

Lao
The Coded Character Set Identifiers (CCSIDs) for Lao conversions are listed.

Latin-1 and Latin-9
The Coded Character Set Identifiers (CCSIDs) for Latin-1 and Latin-9 conversions are listed.

Latin-2
The Coded Character Set Identifiers (CCSIDs) for Latin-2 conversions are listed.

Latin-5

The Coded Character Set Identifiers (CCSIDs) for Latin-5 conversions are listed.

Thai
The Coded Character Set Identifiers (CCSIDs) for Thai conversions are listed.

Traditional Chinese
The Coded Character Set Identifiers (CCSIDs) for Traditional Chinese conversions are listed.

Urdu
The Coded Character Set Identifiers (CCSIDs) for Urdu conversions are listed.

Vietnamese
The Coded Character Set Identifiers (CCSIDs) for Vietnamese conversions are listed.

Unicode data
CICS Transaction Server for z/OS provides limited support for Unicode-encoded character data. Workstations can share UCS-2 or UTF-8 encoded data with CICS Transaction Server for z/OS provided that no conversion is required.

Thai

The Coded Character Set Identifiers (CCSIDs) for Thai conversions are listed.

Table 63. Thai, Client CCSIDs

CLINTCP	CCSID	CPGID	IANA character set name	Comments
1161	01161	01161		PC data: Thai with euro
1162	01162	01162		MS Windows: Thai with euro
9066	09066	00874		PC data: Thai extended SBCS

Table 64. Thai, Server CCSIDs

SRVERCP	CCSID	CPGID	IANA character set name	Comments
1160	01160	01160		Host: Thai with euro
9030	09030	00838		Host: Thai extended SBCS

Related reference

Arabic
The Coded Character Set Identifiers (CCSIDs) for Arabic conversions are listed.

Baltic Rim
The Coded Character Set Identifiers (CCSIDs) for Baltic Rim conversions are listed.

Cyrillic
The Coded Character Set Identifiers (CCSIDs) for Cyrillic conversions are listed.

Devanagari
The Coded Character Set Identifiers (CCSIDs) for Devanagari conversions are listed.

Farsi
The Coded Character Set Identifiers (CCSIDs) for Farsi conversions are listed.

Greek
The Coded Character Set Identifiers (CCSIDs) for Greek conversions are listed.

Hebrew
The Coded Character Set Identifiers (CCSIDs) for Hebrew conversions are listed.

Japanese
The Coded Character Set Identifiers (CCSIDs) for Japanese conversions are listed.

Korean

The Coded Character Set Identifiers (CCSIDs) for Korean conversions are listed.

Lao
The Coded Character Set Identifiers (CCSIDs) for Lao conversions are listed.

Latin-1 and Latin-9
The Coded Character Set Identifiers (CCSIDs) for Latin-1 and Latin-9 conversions are listed.

Latin-2
The Coded Character Set Identifiers (CCSIDs) for Latin-2 conversions are listed.

Latin-5
The Coded Character Set Identifiers (CCSIDs) for Latin-5 conversions are listed.

Simplified Chinese
The Coded Character Set Identifiers (CCSIDs) for Simplified Chinese conversions are listed.

Traditional Chinese
The Coded Character Set Identifiers (CCSIDs) for Traditional Chinese conversions are listed.

Urdu
The Coded Character Set Identifiers (CCSIDs) for Urdu conversions are listed.

Vietnamese
The Coded Character Set Identifiers (CCSIDs) for Vietnamese conversions are listed.

Unicode data
CICS Transaction Server for z/OS provides limited support for Unicode-encoded character data. Workstations can share UCS-2 or UTF-8 encoded data with CICS Transaction Server for z/OS provided that no conversion is required.

Traditional Chinese

The Coded Character Set Identifiers (CCSIDs) for Traditional Chinese conversions are listed.

Table 65. Traditional Chinese, Client CCSIDs

CLINTCP	CCSID	CPGID	IANA character set name	Comments
938	00938	1. 00904 2. 00927		1. PC data: SBCS 2. PC data: DBCS including 6204 user-defined characters
948	00948	1. 01043 2. 00927		1. PC data: Extended SBCS 2. PC data: DBCS including 6204 user-defined characters
950 BIG5	00950	1. 01114 2. 00947	big5	1. PC data: SBCS (IBM BIG5) 2. PC data: DBCS including 13493 CNS, 566 IBM selected, 6204 user-defined characters
964 EUCTW	00964	1. 00367 2. 00960 3. 00961		1. G0: ASCII 2. G1: CNS 11643 plane 1 3. G1: CNS 11643 plane 2
1370	01370	1. 01114 2. 00947		1. PC data: Extended SBCS; with euro 2. PC data: DBCS including 6204 user-defined characters; with euro

Table 66. Traditional Chinese, Server CCSIDs				
SRVERCP	**CCSID**	**CPGID**	**IANA character set name**	**Comments**
937	00937	1. 00037 2. 00835		1. Host: Extended SBCS 2. Host: DBCS including 6204 user-defined characters
1371	01371	1. 01159 2. 00835		1. Host: Extended SBCS; with euro 2. Host: DBCS including 6204 user-defined characters; with euro

Related reference

Arabic
The Coded Character Set Identifiers (CCSIDs) for Arabic conversions are listed.

Baltic Rim
The Coded Character Set Identifiers (CCSIDs) for Baltic Rim conversions are listed.

Cyrillic
The Coded Character Set Identifiers (CCSIDs) for Cyrillic conversions are listed.

Devanagari
The Coded Character Set Identifiers (CCSIDs) for Devanagari conversions are listed.

Farsi
The Coded Character Set Identifiers (CCSIDs) for Farsi conversions are listed.

Greek
The Coded Character Set Identifiers (CCSIDs) for Greek conversions are listed.

Hebrew
The Coded Character Set Identifiers (CCSIDs) for Hebrew conversions are listed.

Japanese
The Coded Character Set Identifiers (CCSIDs) for Japanese conversions are listed.

Korean
The Coded Character Set Identifiers (CCSIDs) for Korean conversions are listed.

Lao
The Coded Character Set Identifiers (CCSIDs) for Lao conversions are listed.

Latin-1 and Latin-9
The Coded Character Set Identifiers (CCSIDs) for Latin-1 and Latin-9 conversions are listed.

Latin-2
The Coded Character Set Identifiers (CCSIDs) for Latin-2 conversions are listed.

Latin-5
The Coded Character Set Identifiers (CCSIDs) for Latin-5 conversions are listed.

Simplified Chinese
The Coded Character Set Identifiers (CCSIDs) for Simplified Chinese conversions are listed.

Thai
The Coded Character Set Identifiers (CCSIDs) for Thai conversions are listed.

Urdu
The Coded Character Set Identifiers (CCSIDs) for Urdu conversions are listed.

Vietnamese
The Coded Character Set Identifiers (CCSIDs) for Vietnamese conversions are listed.

Unicode data

CICS Transaction Server for z/OS provides limited support for Unicode-encoded character data. Workstations can share UCS-2 or UTF-8 encoded data with CICS Transaction Server for z/OS provided that no conversion is required.

Urdu

The Coded Character Set Identifiers (CCSIDs) for Urdu conversions are listed.

Data conversion does not change the direction of Urdu data.

Table 67. Urdu, Client CCSIDs

CLINTCP	CCSID	CPGID	IANA character set name	Comments
868	00868	00868	ibm868	PC data: Urdu
1006	01006	01006		ISO-8: Urdu

Table 68. Urdu, Server CCSIDs

SRVERCP	CCSID	CPGID	IANA character set name	Comments
918	00918	00918	ibm918	Host: Urdu

Related reference

Arabic
The Coded Character Set Identifiers (CCSIDs) for Arabic conversions are listed.

Baltic Rim
The Coded Character Set Identifiers (CCSIDs) for Baltic Rim conversions are listed.

Cyrillic
The Coded Character Set Identifiers (CCSIDs) for Cyrillic conversions are listed.

Devanagari
The Coded Character Set Identifiers (CCSIDs) for Devanagari conversions are listed.

Farsi
The Coded Character Set Identifiers (CCSIDs) for Farsi conversions are listed.

Greek
The Coded Character Set Identifiers (CCSIDs) for Greek conversions are listed.

Hebrew
The Coded Character Set Identifiers (CCSIDs) for Hebrew conversions are listed.

Japanese
The Coded Character Set Identifiers (CCSIDs) for Japanese conversions are listed.

Korean
The Coded Character Set Identifiers (CCSIDs) for Korean conversions are listed.

Lao
The Coded Character Set Identifiers (CCSIDs) for Lao conversions are listed.

Latin-1 and Latin-9
The Coded Character Set Identifiers (CCSIDs) for Latin-1 and Latin-9 conversions are listed.

Latin-2
The Coded Character Set Identifiers (CCSIDs) for Latin-2 conversions are listed.

Latin-5
The Coded Character Set Identifiers (CCSIDs) for Latin-5 conversions are listed.

Simplified Chinese
The Coded Character Set Identifiers (CCSIDs) for Simplified Chinese conversions are listed.

Thai

The Coded Character Set Identifiers (CCSIDs) for Thai conversions are listed.

Traditional Chinese
The Coded Character Set Identifiers (CCSIDs) for Traditional Chinese conversions are listed.

Vietnamese
The Coded Character Set Identifiers (CCSIDs) for Vietnamese conversions are listed.

Unicode data
CICS Transaction Server for z/OS provides limited support for Unicode-encoded character data. Workstations can share UCS-2 or UTF-8 encoded data with CICS Transaction Server for z/OS provided that no conversion is required.

Vietnamese

The Coded Character Set Identifiers (CCSIDs) for Vietnamese conversions are listed.

Table 69. Vietnamese, Client CCSIDs

CLINTCP	CCSID	CPGID	IANA character set name	Comments
1129	01129	01129		ISO-8: Vietnamese
1163	01163	01163		ISO-8: Vietnamese with euro
1258	01258	01258	windows-1258	MS Windows: Vietnamese
5354	05354	01258		MS Windows: Vietnamese, version 2 with euro

Table 70. Vietnamese, Server CCSIDs

SRVERCP	CCSID	CPGID	IANA character set name	Comments
1130	01130	01130		Host: Vietnamese
1164	01164	01164		Host: Vietnamese with euro

Related reference

Arabic
The Coded Character Set Identifiers (CCSIDs) for Arabic conversions are listed.

Baltic Rim
The Coded Character Set Identifiers (CCSIDs) for Baltic Rim conversions are listed.

Cyrillic
The Coded Character Set Identifiers (CCSIDs) for Cyrillic conversions are listed.

Devanagari
The Coded Character Set Identifiers (CCSIDs) for Devanagari conversions are listed.

Farsi
The Coded Character Set Identifiers (CCSIDs) for Farsi conversions are listed.

Greek
The Coded Character Set Identifiers (CCSIDs) for Greek conversions are listed.

Hebrew
The Coded Character Set Identifiers (CCSIDs) for Hebrew conversions are listed.

Japanese
The Coded Character Set Identifiers (CCSIDs) for Japanese conversions are listed.

Korean
The Coded Character Set Identifiers (CCSIDs) for Korean conversions are listed.

Lao

The Coded Character Set Identifiers (CCSIDs) for Lao conversions are listed.

Latin-1 and Latin-9
The Coded Character Set Identifiers (CCSIDs) for Latin-1 and Latin-9 conversions are listed.

Latin-2
The Coded Character Set Identifiers (CCSIDs) for Latin-2 conversions are listed.

Latin-5
The Coded Character Set Identifiers (CCSIDs) for Latin-5 conversions are listed.

Simplified Chinese
The Coded Character Set Identifiers (CCSIDs) for Simplified Chinese conversions are listed.

Thai
The Coded Character Set Identifiers (CCSIDs) for Thai conversions are listed.

Traditional Chinese
The Coded Character Set Identifiers (CCSIDs) for Traditional Chinese conversions are listed.

Urdu
The Coded Character Set Identifiers (CCSIDs) for Urdu conversions are listed.

Unicode data
CICS Transaction Server for z/OS provides limited support for Unicode-encoded character data. Workstations can share UCS-2 or UTF-8 encoded data with CICS Transaction Server for z/OS provided that no conversion is required.

Unicode data

CICS Transaction Server for z/OS provides limited support for Unicode-encoded character data. Workstations can share UCS-2 or UTF-8 encoded data with CICS Transaction Server for z/OS provided that no conversion is required.

More extensive support for conversion to and from Unicode data is available in CICS if you use channels to communicate your data. See Transferring data between programs using channels.

Table 71. Unicode

CLINTCP **SRVERCP**	**CCSID**	**CPGID**	**IANA character set name**	**Comments**
1200 UCS-2	01200	01400	utf-16	Unicode with character set 65535. In the absence of a byte-order mark (BOM), assumed to be UTF-16 BE (big-endian).
1208 UTF-8	01208	01400	utf-8	Unicode with character set 65535. UTF-8.
13488	13488	01400	iso-10646-ucs-2	Unicode with character set 3001 (fixed at Unicode 2.0 character repertoire). In the absence of a byte-order mark, assumed to be UTF16-BE (big-endian).
17584	17584	01400		Unicode with character set 3004 (fixed at Unicode 3.0 character repertoire). in the absence of a byte-order mark, assumed to be UTF16-BE (big-endian).

Related reference

Arabic

The Coded Character Set Identifiers (CCSIDs) for Arabic conversions are listed.

Baltic Rim
The Coded Character Set Identifiers (CCSIDs) for Baltic Rim conversions are listed.

Cyrillic
The Coded Character Set Identifiers (CCSIDs) for Cyrillic conversions are listed.

Devanagari
The Coded Character Set Identifiers (CCSIDs) for Devanagari conversions are listed.

Farsi
The Coded Character Set Identifiers (CCSIDs) for Farsi conversions are listed.

Greek
The Coded Character Set Identifiers (CCSIDs) for Greek conversions are listed.

Hebrew
The Coded Character Set Identifiers (CCSIDs) for Hebrew conversions are listed.

Japanese
The Coded Character Set Identifiers (CCSIDs) for Japanese conversions are listed.

Korean
The Coded Character Set Identifiers (CCSIDs) for Korean conversions are listed.

Lao
The Coded Character Set Identifiers (CCSIDs) for Lao conversions are listed.

Latin-1 and Latin-9
The Coded Character Set Identifiers (CCSIDs) for Latin-1 and Latin-9 conversions are listed.

Latin-2
The Coded Character Set Identifiers (CCSIDs) for Latin-2 conversions are listed.

Latin-5
The Coded Character Set Identifiers (CCSIDs) for Latin-5 conversions are listed.

Simplified Chinese
The Coded Character Set Identifiers (CCSIDs) for Simplified Chinese conversions are listed.

Thai
The Coded Character Set Identifiers (CCSIDs) for Thai conversions are listed.

Traditional Chinese
The Coded Character Set Identifiers (CCSIDs) for Traditional Chinese conversions are listed.

Urdu
The Coded Character Set Identifiers (CCSIDs) for Urdu conversions are listed.

Vietnamese
The Coded Character Set Identifiers (CCSIDs) for Vietnamese conversions are listed.

Appendix B. The conversion process

This section describes in more detail how data conversion works in CICS.

Components

The CICS or user-supplied mirror transactions convert the data, using DFHCNV, DFHCCNV, and the user-replaceable conversion program, DFHUCNV.

DFHCNV

The conversion table. For each resource for which conversion is required, DFHCNV contains a *conversion template*. A conversion template is a table entry defining fields in a data area that are to be converted, and the conversion method to be applied to each field.

You define the DFHCNV table with the DFHCNV resource definition macros described in Defining the conversion table.

DFHCCNV

The CICS program that drives the conversion process. DFHCCNV uses the DFHCNV table to determine the required conversions. It applies standard conversion to those fields in the conversion templates for which nonstandard, user-handled conversion is not specified.

The user-replaceable conversion program, DFHUCNV

A user-replaceable program that allows you to override the standard conversions applied by CICS. You can use it to apply your own conversion logic to specific data fields. (How to do this is described in "User/CICS conversion" on page 340.)

You can use the supplied program as a model on which to base your own version.

You can provide *either*:

- Your own, customized, version of DFHUCNV, *or*
- One or more differently-named conversion programs

Process

This section describes the standard conversions that can be applied by DFHCCNV to specific fields in a conversion template. Other types of conversion are possible, if you write a DFHUCNV program.

Character data

Character data can be converted:

- From ASCII to EBCDIC, on receipt of a request from a connected system, before invoking the EXEC interface
- From EBCDIC to ASCII, on return from the EXEC interface, before the response is transmitted.

The translation tables shipped with CICS conform to the standards described in Character Data Representation Architecture.

Binary data

Binary data can be converted:

- From little-endian to big-endian format, on receipt of a request from a connected system.
- From big-endian to little-endian format, before the response is transmitted.

Standard and nonstandard conversion

There are three ways a single resource, for example a file, can be converted.

- CICS-only conversion—all data fields are handled by the standard CICS conversion program, DFHCCNV

- User/CICS conversion—a combination of nonstandard and standard conversion, in which some data fields are handled by code in the user's conversion program and some by DFHCCNV
- User-only conversion—all data fields are handled by the user's conversion program.

CICS-only conversion

Use CICS-only conversion when the resource contains no data fields that require nonstandard conversion; all can be converted by standard means.

Procedure

1. Create a conversion template, using the DFHCNV macros described in Defining the conversion table.

   This enables DFHCCNV to handle the resource.
2. Specify USREXIT=NO on the DFHCNV TYPE=ENTRY macro that defines the resource.

   This prevents DFHUCNV from being called unnecessarily. Do not specify DATATYP=USERDATA on any of the DFHCNV TYPE=FIELD macros that define the data fields.

User/CICS conversion

Use user/CICS conversion when the resource contains some fields that can be converted by standard means, and some that require nonstandard conversion.

Procedure

1. Create a conversion template.
2. Specify the USREXIT keyword on the DFHCNV TYPE=ENTRY macro that defines the resource.
   - If you specify USREXIT=YES, CICS calls DFHUCNV to convert the data.
   - If you specify USREXIT=*program*, CICS calls the named program to convert the data.
3. Specify DATATYP=USERDATA on the DFHCNV TYPE=FIELD macros that define the nonstandard data fields.

   a) Optional: Define nonstandard fields with a USRTYPE value in the range X'50' through X'80'

      These values are passed to your user program, and can be used to distinguish between different types of nonstandard field.
4. Define standard fields as DATATYP=CHARACTER, PD, BINARY, GRAPHIC, or NUMERIC, as appropriate.
5. Supply a user-written version of DFHUCNV or a differently-named conversion program to handle the nonstandard fields.

   The user-replaceable conversion program gives a description and listing of DFHUCNV, with guidance on how to use it as a basis for your own conversion program.

User-only conversion

The resource contains no fields that can be converted by standard means; all require nonstandard conversion. There are two methods of enabling user-only conversion.

Procedure

1. Create a conversion template.
2. Specify the USREXIT keyword on the DFHCNV TYPE=ENTRY macro that defines the resource.
   - If you specify USREXIT=YES, CICS calls DFHUCNV to convert the data.
   - If you specify USREXIT=*program*, CICS calls the named program to convert the data.
3. Specify DATATYP=USERDATA on the DFHCNV TYPE=FIELD macros that define the nonstandard data fields.

   a) Optional: Define nonstandard fields with a USRTYPE value in the range X'50' through X'80'

These values are passed to your user program, and can be used to distinguish between different types of nonstandard field.

4. Supply a user-written version of DFHUCNV or a differently-named conversion program to handle all fields.

   The user-replaceable conversion program gives a description and listing of DFHUCNV, with guidance on how to use it as a basis for your own conversion program.

5.

Sequence of conversion processing

This is the sequence of conversion processing.

1. Unless USREXIT=NO is specified in the DFHCNV TYPE=ENTRY macro that defines the conversion template for the resource, DFHCCNV links to DFHUCNV, passing the parameter list described in Parameter list (DFHUVNDS).

   Note:

   a. If you have not defined a template, DFHUCNV is invoked, on the assumption that the user program is to handle all conversions for the resource.

   b. DFHUCNV must be present in your system unless all DFHCNV TYPE=ENTRY macros specify USREXIT=NO.

2. If a conversion template is defined for the resource, DFHUCNV is responsible for converting any fields with a type in the user-data range.

   If no conversion template is defined for the resource, DFHUCNV is responsible for determining the format of the data, and for converting all appropriate fields.

3. On return from DFHUCNV, DFHCCNV carries out any standard conversions specified in the conversion template for fields that are not subject to user-defined conversion.

4. The shipped request is executed.

Figure 91 on page 342 illustrates the conversion process.

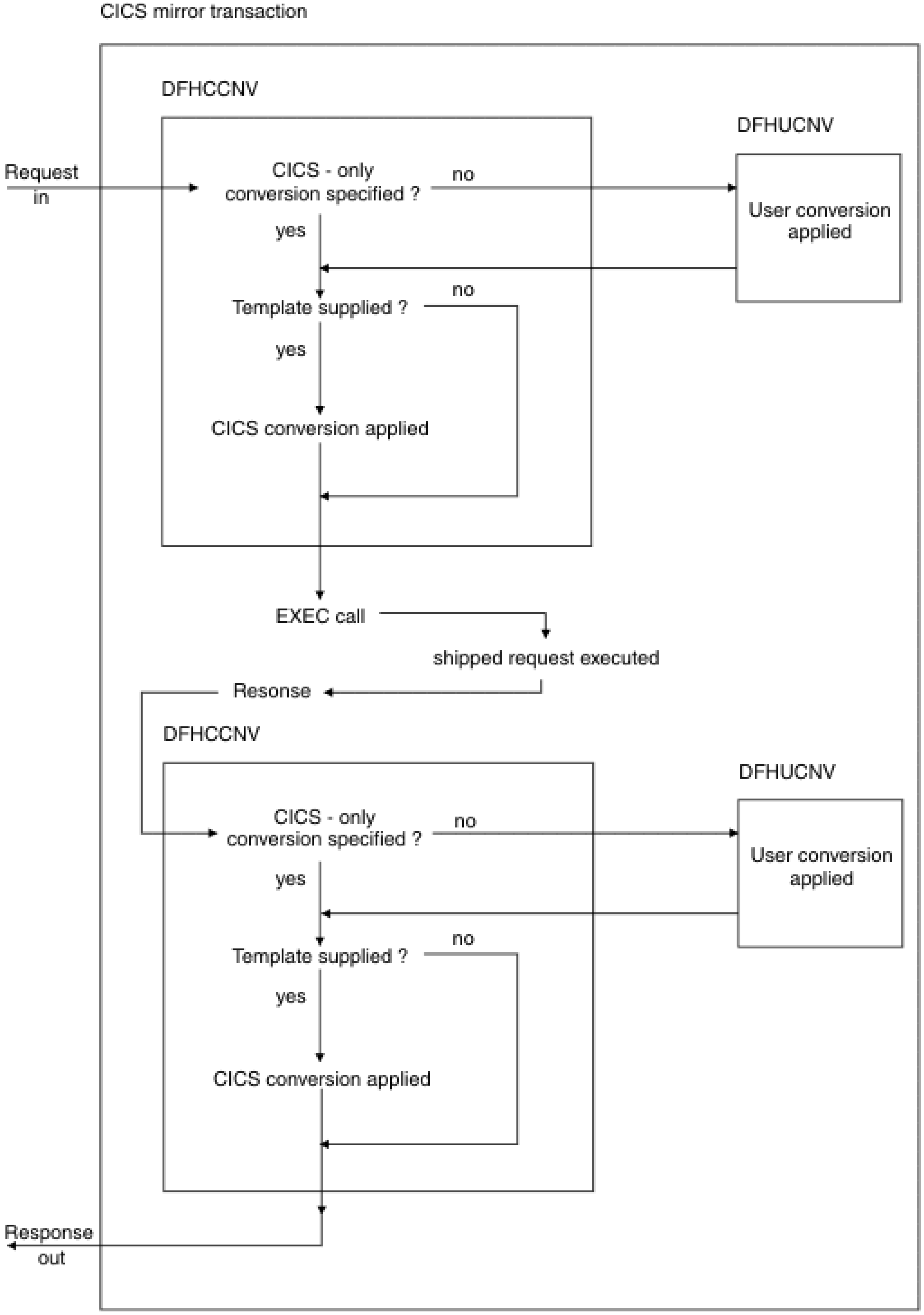


Figure 91. The data conversion process

Appendix C. Intercommunication rules and restrictions checklist

This appendix provides a checklist of the rules and restrictions that apply to intersystem communication and multiregion operation.

Most of these rules and restrictions also appear in the body of the book. The rules apply to:

Transaction routing

There are a number of rules and restrictions that apply to transaction routing, to review the entire checklist see "Transaction routing" on page 344.

Dynamic routing of DPL requests

For a distributed program link request to be eligible for dynamic routing, the remote program must either be defined to the local system as DYNAMIC, or not be defined to the local system. Daisy-chaining of dynamically-routed DPL requests is not supported, see Daisy-chaining of DPL requests.

Automatic transaction initiation

- A terminal-associated transaction that is initiated by the transient data trigger level facility must reside on the same system as the transient data queue that causes its initiation. This restriction applies to both macro-level and command-level application programs.
- There are restrictions on the dynamic routing of transactions initiated by EXEC CICS START commands, see the list of conditions in "Transaction routing" on page 344.

Basic mapping support

- BMS support must reside on each system that owns a terminal through which paging commands can be entered.
- A BMS ROUTE request cannot be used to send a message to a selected remote operator or operator class unless the terminal at which the message is to be delivered is specified in the route list.

Acquiring LUTYPE6.1 sessions

- If an application tries to acquire an LUTYPE6.1 connection and the remote system is unavailable, the connection is placed out of service.
- If the remote system is a CICS region that uses AUTOCONNECT, the connection is placed back in service when the initialization of the remote system is complete
- Otherwise, you must manually place the connection back in service.

Function shipping

You can use CICS function shipping to write CICS application programs without regard to the location of the requested resources. They use file control commands, temporary-storage commands, and other functions in the same way.

Syncpointing

SYNCPOINT ROLLBACK commands are supported by APPC, IPIC, and MRO sessions.

Local and remote names

Local names are translated to remote names according to these rules.

- Transaction identifiers are translated from local names to remote names when a request to execute a transaction is transmitted from one CICS system to another. However, a transaction identifier specified in an EXEC CICS RETURN command is not translated when it is transmitted from the application-owning region to the terminal-owning region.
- Terminal identifiers are translated from local names to remote names when a transaction routing request to execute a transaction on a specified terminal is shipped from one CICS system to another. However if an EXEC CICS START command specifying a terminal identification is function shipped from one CICS system to another, the terminal identification is not translated from local name to remote name.

Main terminal transaction

Only locally owned terminals can be queried and modified by the main terminal transaction CEMT. The only terminals visible to this transaction are those owned by the system on which the main terminal transaction is running.

Installation and operations

- Module DFHIRP must be made LPA-resident; otherwise jobs and console commands may abend on completion.
- Interregion communication requires subsystem interface (SSI) support.
- Do not install more than one APPC connection between an LU-LU pair.
- Do not install an APPC and an LUTYPE6.1 connection at the same time between an LU-LU pair.
- Do not install more than one MRO connection between the same two CICS regions.
- Do not install more than one generic EXCI connection on a CICS region.

Customization

- Communication between node error programs, user exits, and user programs is the responsibility of the user.
- Transactions that recover input messages for protected tasks after a system crash must run on the same system as the terminal that invoked the protected task.

MRO abend codes

- An IRC transaction in send state is unable to receive an error reason code if its partner has to abend. It abends itself with code AZI2, which should be interpreted as a general indication that the other side is no longer there. The real reason for the failure can be read from the CSMT destination of the CICS region that first detected the error. For example, a security violation in attaching a back-end transaction is reported as such by the front end only if the initiating command is CONVERSE and not SEND.

Transaction routing

Review this checklist of the rules and restrictions that apply to transaction routing.

- A transaction routing path between a terminal and a transaction must not turn back on itself. For example, if system A specifies that a transaction is on system B, system B specifies that it is on system C, and system C specifies that it is on system A, the attempt to use the transaction from system A is abended when system C tries to route back to system A.

  This restriction also applies if the routing transaction, CRTE, is used to establish all or part of a path that turns back on itself.
- Transaction routing that uses the following “terminals” is not supported:

  - LUTYPE6.1 sessions.
  - IPIC sessions.
  - MRO sessions.
  - IBM 7770 and 2260 terminals.
  - Pipeline logical units with pooling.
  - MVS system consoles. Messages entered through a console can be directed to any CICS system using the **MODIFY** command.
- The transaction CEOT is not supported by the transaction routing facility.
- The execution diagnostic facility (EDF) can be used in single-terminal mode to test a remote transaction.

  EDF running in two-terminal mode is supported only when both of the terminals and the user transaction are on the same system; that is, when no transaction routing is involved.

  Sending EDF information over an IPIC connection is supported between CICS TS 5.1 or later regions. For CICS TS 4.2 and earlier releases, when using an IPIC connection, use CEDX in the remote region for transactions that are defined in the terminal owning region (TOR) as remote.
- The user area of the terminal control table terminal entry (TCTTE) is updated at task-attach and task-detach times. Therefore, a user exit program running on the terminal-owning region and examining the user area while the terminal is running a remote transaction does not necessarily see the same values as a user exit running at the same time in the application-owning region. Note also that the user areas must be defined as having the same length in both systems.
- All programs, tables, and maps that are used by a transaction must be on the system that owns the transaction. The programs, tables, and maps can be duplicated in as many systems as necessary.
- When routing transactions to or from APPC devices, CICS does not support CPI Communications conversations with sync level characteristics of CM_SYNC_POINT.
- Terminal control table user areas (TCTUAs) are not shipped when the principal facility is an APPC parallel session.
- For a transaction started by a terminal-related **EXEC CICS START** command to be eligible for enhanced routing, *all* of the following conditions must be met:
  - The START command is a member of the subset of eligible START commands; that is, it meets all the following conditions:
    - The START command specifies the TERMID option, which names the principal facility of the task that issues the command; that is, the transaction to be started must be terminal-related, and associated with the principal facility of the starting task.
    - The principal facility of the task that issues the START command is *not* a surrogate client virtual terminal.
    - The SYSID option of the START command does not specify the name of a remote region; that is, the remote region on which the transaction is to be started must not be specified explicitly.
  - The requesting region and the TOR, if they are different, are connected by one of the following links:
    - An MRO link
    - An APPC parallel-session link
    - An IPIC link
  - The TOR and the target region are connected by one of the following links:
    - An MRO link.
    - An APPC single- or parallel-session link. If an APPC link is used, at least one of the following conditions must be met:
      1. Terminal-initiated transaction routing has previously taken place over the link.
      2. CICSPlex SM is being used for routing.

    - An IPIC link.
  - The transaction definition in the requesting region specifies ROUTABLE(YES).
  - If the transaction is to be routed dynamically, the transaction definition in the TOR specifies DYNAMIC(YES).

  For more information about enhanced routing, see Routing transactions invoked by START commands.

- For a non-terminal-related START request to be eligible for enhanced routing, all of the following conditions must be met:
  - The requesting region and the target region are connected by one of the following links:
    - An MRO link.
    - An APPC single- or parallel-session link. If an APPC link is used, and the distributed routing program is to be called on the target region, at least one of the following conditions must be met:
      1. Terminal-initiated transaction routing has previously taken place over the link.
      2. CICSPlex SM is being used for routing.
    - An IPIC link.
  - The transaction definition in the requesting region specifies ROUTABLE(YES).
  - If the request is to be routed dynamically, the following conditions must be met:
    - The transaction definition in the requesting region specifies DYNAMIC(YES).
    - The SYSID option of the START command does not specify the name of a remote region; that is, the remote region on which the transaction is to be started must not be specified explicitly.

  For more information about enhanced routing, see Routing transactions invoked by START commands.

- The following types of dynamic transaction routing requests cannot be daisy-chained:
  - Non-terminal-related START requests
  - CICS business transaction services processes and activities

Appendix D. CICS mapping to the APPC architecture

This appendix shows how the APPC programming language is implemented by CICS.

The APPC programming language is described in the SNA publication *Transaction Programmer's Reference Manual for LU Type 6.2*.

Supported option sets

This table shows which APPC option sets are supported by CICS and which are not.

Table 72. CICS support of APPC options sets

Set #	Set name	Supported
101	Clear the LU's send buffer	Yes
102	Get attributes	Yes
103	Post on receipt with test for posting	No
104	Post on receipt with wait	No
105	Prepare to receive	Yes
106	Receive immediate **Note:** CICS programs support receive_immediate requests provided these requests are coded using the common programming Interface for communications.	Yes
108	Sync point services	Yes
109	Get TP name and instance identifier	No
110	Get conversation type	Yes
111	Recovery from program errors detected during syncpoint	Yes
201	Queued allocation of a contention-winner session	No
203	Immediate allocation of a session	Yes
204	Conversations between programs located at the same LU	No
211	Session-level LU-LU verification	Yes
212	User ID verification	Yes
213	Program-supplied user ID and password	No
214	User ID authorization	Yes
215	Profile verification and authorization	Yes
217	Profile pass-through	No
218	Program-supplied profile	No
241	Send PIP data	Yes
242	Receive PIP data	Yes
243	Accounting	Yes
244	Long locks	No

Table 72. CICS support of APPC options sets (continued)

Set #	Set name	Supported
245	Test for request-to-send received	Yes
246	Data mapping	No
247	FMH data	No
249	Vote read-only response to a syncpoint operation	No
251	Extract transaction and conversation identity information	No
290	Logging of data in a system log	No
291	Mapped conversation LU services component	Yes
401	Reliable one-way brackets	No
501	CHANGE_SESSION_LIMIT verb	Yes
502	ACTIVATE_SESSION verb	Yes
504	DEACTIVATE_SESSION verb	No
505	LU-definition verbs	Yes
601	MIN_CONWINNERS_TARGET parameter	No
602	RESPONSIBLE(TARGET) parameter	No
603	DRAIN_TARGET(NO) parameter	No
604	FORCE parameter	No
605	LU-LU session limit	No
606	Locally known LU names	Yes
607	Uninterpreted LU names	No
608	Single-session reinitiation	No
610	Maximum RU size bounds	Yes
611	Session-level mandatory cryptography	No
612	Contention-winner automatic activation limit	No
613	Local maximum (LU, mode) session limit	Yes
616	CPSVCMG modename support	No
617	Session-level selective cryptography	No

CICS implementation of control operator verbs

CICS supports control operator verbs in a variety of ways.

Some verbs are supported by the CICS main terminal transaction CEMT. The relevant CEMT commands are:

- **`CEMT INQUIRE CONNECTION`**
- **`CEMT SET CONNECTION`**
- **`CEMT INQUIRE MODENAME`**
- **`CEMT SET MODENAME`**

Tip: In CICS Explorer, the ISC/MRO connections operations view provides a functional equivalent to the INQUIRE and SET CONNECTION commands. See ISC/MRO Connections view in the CICS Explorer product documentation.

CEMT is normally entered by an operator at a display device. It is described in CEMT - main terminal.

The inquire and set operations for connections and modenames are also available at the CICS SPI, using the following commands:

- **`EXEC CICS INQUIRE CONNECTION`**
- **`EXEC CICS SET CONNECTION`**
- **`EXEC CICS INQUIRE MODENAME`**
- **`EXEC CICS SET MODENAME`**

Programming information about these commands is given in INQUIRE CONNECTION.

Some control operator verbs are supported by CICS resource definition. The definition of APPC links is described in Defining APPC links.

You can change some CONNECTION and SESSION attributes while CICS is running by discarding the resource and creating a new one.

Control operator verbs

The way in which CICS implements APPC control operator verbs is shown in a set of tables.

See "Return codes for control operator verbs" on page 356 for details of the corresponding return code mapping.

Note: Wherever CEMT is shown, the equivalent form of EXEC CICS command can be used.

Tip: In CICS Explorer, the **ISC/MRO Connections** view provides a functional equivalent to the SET and INQUIRE CONNECTION commands. The **Terminals**, **Local Transactions**, and **Remote Transactions** views provide functional equivalents to the INQUIRE TERMINAL and INQUIRE TRANSACTION commands, respectively. See SM Operations views in the CICS Explorer product documentation.

Table 73. CHANGE_SESSION_LIMIT

CHANGE_SESSION_LIMIT	CEMT SET MODENAME
LU_NAME(vble)	CONNECTION( )
MODE_NAME(vble)	MODENAME( )
LU_MODE_SESSION_LIMIT(vble)	AVAILABLE( )
MIN_CONWINNERS_SOURCE(vble)	CICS negotiates a revised value, based on the AVAILABLE request and the MAXIMUM attribute of the SESSIONS resource.
MIN_CONWINNERS_TARGET(vnle)	Not supported.
RESPONSIBLE(source)	Yes.
RESPONSIBLE(target)	Not supported. CICS does not support receipt of RESP(TARGET).
RETURN_CODE	Supported.

Table 74. INITIALIZE_SESSION_LIMIT

INITIALIZE_SESSION_LIMIT	Specified in SESSIONS resource
LU_NAME(vble)	CONNECTION( )
MODE_NAME(vble)	MODENAME( )

Table 74. INITIALIZE_SESSION_LIMIT (continued)

INITIALIZE_SESSION_LIMIT	**Specified in SESSIONS resource**
LU_MODE_SESSION_LIMIT(vble)	MAXIMUM(value1,)
MIN_CONWINNERS_SOURCE(vble)	MAXIMUM(,value2)
MIN_CONWINNERS_TARGET(vnle)	Not supported.
RETURN_CODE	Supported.

Table 75. PROCESS_SESSION_LIMIT

PROCESS_SESSION_LIMIT	**Automatic action by CICS-supplied transaction CLS1 when CNOS is received by a target CICS system.**
RESOURCE(vble)	Connection resource.
LU_NAME(vble)	Passed internally.
MODE_NAME(vble1,vble2)	Passed internally.
RETURN_CODE	Supported.

Table 76. RESET_SESSION_LIMIT

RESET_SESSION_LIMIT	**CEMT SET MODENAME (for individual modegroups) or CEMT SET CONNECTION RELEASED (to reset all modegroups)**
LU_NAME(vble)	CONNECTION( )
MODE_NAME(ALL)	SET CONNECTION( ) RELEASED
MODE_NAME(ONE(vble))	MODENAME( ) AVAILABLE(0)
MODE_NAME(ONE('SNASVCMG'))	SET CONNECTION( ) RELEASED
RESPONSIBLE(SOURCE)	Yes.
RESPONSIBLE(TARGET)	Not supported.
DRAIN_SOURCE(NO|YES)	CICS supports YES.
DRAIN_TARGET(NO|YES)	CICS supports YES.
FORCE(NO|YES)	Not supported.
RETURN_CODE	Supported.

Table 77. ACTIVATE_SESSION

ACTIVATE_SESSION	**CEMT SET MODENAME ACQUIRED (for individual modegroups) or CEMT SET CONNECTION ACQUIRED (for SNASVCMG sessions)**
LU_NAME(vble)	CONNECTION( )
MODE_NAME(vble)	MODENAME( ) ACQUIRED
MODE_NAME('SNASVCMG')	Activated when CEMT SET CONNECTION ACQUIRED is issued.
RETURN_CODE	Supported.

Table 78. DEACTIVATE_CONVERSATION_GROUP

DEACTIVATE_CONVERSATION_GROUP	Not supported.

Table 79. DEACTIVATE_SESSION

DEACTIVATE_SESSION	Not supported.

Table 80. DEFINE_LOCAL_LU

DEFINE_LOCAL_LU	SESSION resource and system initialization parameters
FULLY_QUALIFIED_LU_NAME(vble)	Cannot be specified. CICS uses the network LU name (APPLID on DFHSIT).
LU_SESSION_LIMIT(NONE)	Not supported.
LU_SESSION_LIMIT(VALUE(vble))	Total of MAX(nn) on all sessions.
SECURITY(ADD USER_ID(vble))	In an external security manager (ESM).
SECURITY(ADD PASSWORD(vble))	Not supported; defined in an ESM.
SECURITY(ADD PROFILE(vble))	Not supported; defined in an ESM.
SECURITY(DELETE USER_ID(vble))	Supported in an ESM.
SECURITY(DELETE PASSWORD(vble))	Not supported; defined in an ESM.
MAP_NAME(ADD(vble))	Not supported.
MAP_NAME(DELETE(vble))	Not supported.
BIND_RSP_QUEUE_CAPACITY(YES|NO)	Not supported.

Table 81. DEFINE_MODE

DEFINE_MODE	EXEC CICS CONNECT PROCESS + MODEENT macro (ACF/Communications Server systems definition) + SESSIONS resource
FULLY_QUALIFIED_LU_NAME(vble)	Cannot be specified. LU identified via CONNECTION on SESSIONS.
MODE_NAME(vble)	MODENAME on SESSIONS is mapped to LOGMODE on MODEENT.
SEND_MAX_RU_SIZE_LOWER_BOUND (vble)	Fixed at 8.
SEND_MAX_RU_SIZE_UPPER_BOUND (vble)	SENDSIZE on SESSIONS.
PREFERRED_RECEIVE_RU_SIZE (vble)	Not supported.
PREFERRED_SEND_RU_SIZE (vble)	Not supported.
RECEIVE_MAX_RU_SIZE_LOWER _BOUND (vble)	Fixed at 256.
RECEIVE_MAX_RU_SIZE_UPPER _BOUND (vble)	RECEIVESIZE on SESSIONS.
SINGLE_SESSION_REINITIATION OPERATOR	Not supported.
SINGLE_SESSION_REINITIATION PLU	Not supported.
SINGLE_SESSION_REINITIATION SLU	Not supported.

Table 81. DEFINE_MODE (continued)

DEFINE_MODE	EXEC CICS CONNECT PROCESS + MODEENT macro (ACF/Communications Server systems definition) + SESSIONS resource
SINGLE_SESSION_REINITIATION PLU_OR_SLU	Not supported.
SESSION_LEVEL_CRYPTOGRAPHY (NOT_SUPPORTED)	Default.
SESSION_LEVEL_CRYPTOGRAPHY (MANDATORY)	Not supported.
SESSION_LEVEL_CRYPTOGRAPHY (SELECTIVE)	Not supported.
CONWINNER_AUTO_ACTIVATE_LIMIT (vble)	MAXIMUM(,value2) on SESSIONS.
SESSION_DEACTIVATED_TP_NAME (vble)	Not supported.
LOCAL_MAX_SESSION_LIMIT (vble)	MAXIMUM(nn,) on SESSIONS.

Table 82. DEFINE_REMOTE_LU

DEFINE_REMOTE_LU	CONNECTION resource
FULLY_QUALIFIED_LU_NAME(vble)	Cannot be specified.
LOCALLY_KNOWN_LU_NAME(NONE)	Not supported.
LOCALLY_KNOWN_LU_NAME (NAME(vble))	CONNECTION(name)
UNINTERPRETED_LU_NAME(NONE)	Defaults to CONNECTION(name).
UNINTERPRETED_LU_NAME (NAME(vble))	NETNAME on CONNECTION.
INITIATE_TYPE(INITIATE_ONLY)	Not supported.
INITIATE_TYPE(INITIATE_OR_QUEUE)	Not supported.
PARALLEL_SESSION_SUPPORT(YES|NO)	SINGLESESS(NO|YES) on CONNECTION.
CNOS_SUPPORT(YES|NO)	Always YES.
LU_LU_PASSWORD(NONE)	Default on CONNECTION.
LU_LU_PASSWORD(VALUE(vble))	BINDPASSWORD on CONNECTION, or SESSKEY in RACF APPCLU profile.
SECURITY_ACCEPTANCE(NONE)	ATTACHSEC(LOCAL)
SECURITY_ACCEPTANCE (CONVERSATION)	ATTACHSEC(VERIFY)
SECURITY_ACCEPTANCE (ALREADY_VERIFIED)	ATTACHSEC(IDENTIFY) or ATTACHSEC(PERSISTENT).

Table 83. DEFINE_TP

DEFINE_TP	TRANSACTION resource
TP_NAME(vble)	TRANSACTION(name)
STATUS(ENABLED)	STATUS(ENABLED)
STATUS(TEMP_DISABLED)	Not supported.
STATUS(PERM_DISABLED)	STATUS(DISABLED)
CONVERSATION_TYPE(MAPPED|BASIC)	Supported for all TPs (determined by choice of command).

Table 83. DEFINE_TP (continued)

DEFINE_TP	**TRANSACTION resource**
SYNC_LEVEL(NONE|CONFIRMvSYNCPT)	SYNCPT for all TPs (actual level specified on CONNECT PROCESS).
SECURITY_REQUIRED(NONE)	Not supported; defined in an ESM.
SECURITY_REQUIRED(CONVERSATION)	Not supported; defined in an ESM.
SECURITY_REQUIRED (ACCESS(PROFILE))	Not supported.
SECURITY_REQUIRED (ACCESS(USER_ID))	Not supported; defined in an ESM.
SECURITY_REQUIRED (ACCESS(USER_ID_PROFILE))	Not supported.
SECURITY_ACCESS(ADD(USER_ID(vble)))	Transaction can be redefined.
SECURITY_ACCESS(ADD(PROFILE(vble)))	Transaction can be redefined.
SECURITY_ACCESS (DELETE(USER_ID(vble)))	Transaction can be redefined.
SECURITY_ACCESS (DELETE(PROFILE(vble)))	Transaction can be redefined.
PIP(NO)	Specified for all TPs.
PIP(YES(vble))	Specified on CONNECT PROCESS.
PIP(NO_LU_VERIFICATION)	Default for all PIP data.
DATA_MAPPING(NO|YES)	DATA_MAPPING(NO) for all TPs.
FMH_DATA(NO|YES)	FMH_DATA(YES) for all TPs.
PRIVILEGE(NONE)	Not supported.
PRIVILEGE(CNOS)	Not supported.
PRIVILEGE(SESSION_CONTROL)	Not supported.
PRIVILEGE(DEFINE)	Not supported.
PRIVILEGE(DISPLAY)	Not supported.
PRIVILEGE(ALLOCATE_SERVICE_TP)	Not supported.
INSTANCE_LIMIT(vble)	Not supported.
RETURN_CODE	Supported.

Table 84. DELETE

DELETE	**EXEC CICS DISCARD**
LOCAL_LU_NAME(vble)	Not supported.
REMOTE_LU_NAME	Not supported.
MODE_NAME	Not supported.
TP_NAME	DISCARD TRANSACTION( )
RETURN_CODE	Supported.

Table 85. DISPLAY_LOCAL_LU

DISPLAY_LOCAL_LU	**CEMT INQUIRE CONNECTION + CEMT INQUIRE MODENAME + CEMT INQUIRE TRANSACTION**
FULLY_QUALIFIED_LU_NAME(vble)	Cannot be specified in CICS. The APPLID on DFHSIT serves as identifier for the local LU. Specific information can be had by identifying the remote LU. Otherwise, the universal ID * can be used.
LU_SESSION_LIMIT(vble)	MAXIMUM on INQ MODENAME.
LU_SESSION_COUNT(vble)	ACTIVE on INQ MODENAME
SECURITY(vble)	Not available.
MAP_NAMES(vble)	Not supported.
REMOTE_LU_NAMES(vble)	INQ CONNECTION(*)
TP_NAMES(vble)	INQ TRANSACTION(*)
BIND_RSP_QUEUE_CAPABILITY(vble)	Not supported.
RETURN_CODE	Supported.

Table 86. DISPLAY_REMOTE_LU

DISPLAY_REMOTE_LU	**CEMT INQUIRE CONNECTION + CEMT INQUIRE MODENAME**
FULLY_QUALIFIED_LU_NAME(vble)	Cannot be specified; CONNECTION or MODENAME may be used.
LOCALLY_KNOWN_LU_NAME(vble)	CONNECTION name.
UNINTERPRETED_LU_NAME(vble)	NETNAME on INQ CONNECTION.
INITIATE_TYPE(vble)	Not supported.
PARALLEL_SESSION_SUPPORT(vble)	SINGLESESS(Y|N) attribute.
CNOS_SUPPORT(vble)	Always YES.
SECURITY_ACCEPTANCE_LOCAL_LU (vble)	Not available.
SECURITY_ACCEPTANCE_REMOTE_LU (vble)	Not available.
MODE_NAMES(vble)	MODENAME attribute of the SESSIONS resource.
RETURN_CODE	Supported.

Table 87. DISPLAY_MODE

DISPLAY_MODE	**CEMT INQUIRE MODENAME + CEMT INQUIRE TERMINAL**
FULLY_QUALIFIED_LU_NAME(vble)	Cannot be specified.
MODE_NAME(vble)	MODENAME attribute of the SESSIONS resource.
LOCAL_MAX_SESSION_LIMIT(vble)	AVA on CEMT INQ MODENAME.
CONVERSATION_GROUP_IDS(vble)	Not supported.
SEND_MAX_RU_SIZE_LOWER_BOUND (vble)	Fixed at 8.
SEND_MAX_RU_SIZE_UPPER_BOUND (vble)	Not available.

Table 87. DISPLAY_MODE (continued)

DISPLAY_MODE	CEMT INQUIRE MODENAME + CEMT INQUIRE TERMINAL
RECEIVE_MAX_RU_SIZE_LOWER_BOUND (vble)	Fixed at 256.
RECEIVE_MAX_RU_SIZE_UPPER_BOUND (vble)	Not available.
PREFERRED_SEND_RU_SIZE(vble)	Not supported.
PREFERRED_RECEIVE_RU_SIZE(vble)	Not supported.
SINGLE_SESSION_REINITIATION(vble)	Not supported.
SESSION_LEVEL_CRYPTOGRAPHY(vble)	Not available.
SESSION_DEACTIVATED_TP_NAME	Not supported.
CONWINNER_AUTO_ACTIVATE_LIMIT (vble)	Not available.
LU_MODE_SESSION_LIMIT(vble)	MAXIMUM on INQ MODENAME.
MIN_CONWINNERS(vble)	Not supported.
MIN_CONLOSERS(vble)	Not supported.
TERMINATION_COUNT(vble)	Not supported.
DRAIN_LOCAL_LU(vble)	Not supported.
DRAIN_REMOTE_LU(vble)	Not supported.
LU_MODE_SESSION_COUNT(vble)	ACTIVE on INQ MODENAME.
CONWINNERS_SESSION_COUNT(vble)	Not available.
CONLOSERS_SESSION_COUNT(vble)	Not available.
SESSION_IDS(vble)	INQ TERMINAL(*).
RETURN_CODE	Supported.

Table 88. DISPLAY_TP

DISPLAY_TP	CEMT INQUIRE TRANSACTION
TP_NAME(vble)	TRANSACTION(tranid)
STATUS(vble)	ENABLED/DISABLED.
CONVERSATION_TYPE(vble)	CICS TPs allow both types.
SYNC_LEVEL(vble)	CICS TPs allow all sync levels.
SECURITY_REQUIRED(vble)	Not available.
SECURITY_ACCESS(vble)	Not available.
PIP(vble)	CICS TPs allow PIP YES and NO.
DATA_MAPPING(vble)	Always NO.
FMH_DATA(vble)	Always YES.
PRIVILEGE(vble)	Not supported.
INSTANCE_LIMIT(vble)	Not supported.
INSTANCE_COUNT(vble)	CEMT INQ TRAN( )

Table 88. DISPLAY_TP (continued)

DISPLAY_TP	CEMT INQUIRE TRANSACTION
RETURN_CODE	Supported.

Return codes for control operator verbs

When you change the state of a CONNECTION or a MODENAME, the LU services manager starts asynchronously.

Some of the errors that may occur are detected by immediately. Other errors are not detected until a later time, when the LU services manager transaction (CLS1) runs.

If CLS1 detects errors, it causes messages to be written to the CSMT log, as shown in Table 89 on page 356. In normal operation, the CICS main terminal operator may not want to inspect the CSMT log when a command has been issued. So in general, the operator, after issuing a command to change parameters should wait for a few seconds for the request to be carried out and then reissue the INQUIRE version of the command to check that the requested change has been made. In the few cases when an error occurs, the main terminal control operator can refer to the CSMT log.

The message used to report the results of CLS1 execution is DFHZC4900. The explanatory text that accompanies the message varies and is summarized in Table 89 on page 356. In certain cases, DFHZC4901 is also issued to give further information.

Table 89. Messages triggered by CLS1

APPC RETURN CODE	CICS MESSAGE
OK	DFHZC4900 result = SUCCESSFUL
ACTIVATION_FAILURE_RETRY	DFHZC4900 result = VALUES AMENDED + DFHZC4901 MAX = 0
ACTIVATION_FAILURE_NO_RETRY	DFHZC4900 result = VALUES AMENDED + DFHZC4901 MAX = 0
ALLOCATION_ERROR	SYSTEM NOT ACQUIRED is returned to the operator.
COMMAND_RACE_REJECT	DFHZC4900 result = RACE DETECTED
LU_MODE_SESSION_LIMIT_CLOSED	DFHZC4900 result = VALUES AMENDED + DFHZC4901 MAX = 0
LU_MODE_SESSION_LIMIT_EXCEEDED	DFHZC4900 result = VALUES AMENDED + DFHZC4901 MAX = (negotiated value)
LU_MODE_SESSION_LIMIT_NOT_ZERO	DFHZC4900 result = VALUES AMENDED + DFHZC4901 MAX = (negotiated value)
LU_MODE_SESSION_LIMIT_ZERO	DFHZC4900 result = VALUES AMENDED + DFHZC4901 MAX = 0
LU_SESSION_LIMIT_EXCEEDED	DFHZC4900 result = VALUES AMENDED + DFHZC4901 MAX = (negotiated value)
PARAMETER_ERROR	Checked immediately
REQUEST_EXCEEDS_MAX_ALLOWED	Checked immediately
RESOURCE_FAILURE_NO_RETRY	The LU services manager transaction (CLS1) abends with abend code ATNI.
UNRECOGNIZED_MODE_NAME	DFHZC4900 result = MODENAME NOT RECOGNIZED

CICS deviations from APPC architecture

This section describes the way in which the CICS implementation of APPC differs from the architecture described in the *Format and Protocol Reference Manual: Architecture Logic for LU Type 6.2*.

Be aware of the folllwing deviation:

- **CICS implementation**: CICS checks incoming BIND requests for valid combinations of the CNOS indicator (BIND RQ byte 24 bit 6) and the PARALLEL-SESSIONS indicator (BIND RQ byte 24 bit 7). If an incorrect combination is found (that is, PARALLEL-SESSIONS specified but CNOS not specified), CICS sends a negative response to the BIND request.

  APPC architecture: The secondary logical unit (SLU), or BIND request receiver, should negotiate the CNOS and PARALLEL-SESSIONS indicators to the supported level and return them in the BIND response. The SLU should not check for an incorrect combination of these indicators.

APPC transaction routing deviations from APPC architecture

A transaction program cannot use ISSUE SIGNAL while in syncfree, syncsend, or syncreceive state. Attempting to do so may result in a state check. This single deviation applies only to APPC transaction routing.

CICS mapping to the APPC verbs

The APPC verbs are implemented by equivalent CICS application programming commands.

The APPC programming language is described in *Transaction Programmer's Reference Manual for LU Type 6.2*.

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 Appendix D, "CICS mapping to the APPC architecture," on page 347.

Command mapping for APPC basic conversations

The APPC verbs for basic conversations are implemented by equivalent CICS application programming commands.

The following tables show the mapping between APPC verbs and CICS commands for basic conversations. See "Return codes for APPC basic conversations" on page 362 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

ALLOCATE	**EXEC CICS GDS ALLOCATE** **+ EXEC CICS GDS CONNECT PROCESS**
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. 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 around 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

The return codes for APPC basic conversation verbs are indicated by equivalent CICS return codes.

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.

APPC RETURN_CODE	**CICS return codes**
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.
PROG_ERROR_NO_TRUNC	CDBERRCD = 08890000 (RECEIVE Only)
PROG_ERROR_TRUNC	CDBERRCD = 08890001
PROG_ERROR_PURGING	CDBERRCD = 08890000
RESOURCE_FAILURE_RETRY	CDBERRCD = A000
RESOURCE_FAILURE_NO_RETRY	CDBERRCD = A000
SVC_ERROR_NO_TRUNC	CDBERRCD = 08890100 (RECEIVE Only)
SVC_ERROR_TRUNC	CDBERRCD = 08890101
SVC_ERROR_PURGING	CDBERRCD = 08890100

APPC RETURN_CODE	CICS return codes
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 APPC verbs for mapped conversations are implemented by equivalent CICS application programming commands.

See "Return codes for APPC mapped conversations" on page 369 for details of the corresponding return code mapping.

This table has two columns. The headers are in the first row.

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

This table has two columns. The headers are in the first row.

(continued)

MC_ALLOCATE	**EXEC CICS ALLOCATE** **+ EXEC CICS CONNECT PROCESS**
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. 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 around 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, before 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

The return codes for APPC mapped conversation verbs are indicated by equivalent CICS return codes.

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.

APPC RETURN_CODE	CICS return codes
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.
PARAMETER_ERROR (Invalid LU name)	SYSIDERR (EIBRCODE = D0 04 .. or D0 0C ..)
PARAMETER_ERROR (Invalid mode name)	CBIDERR raised for invalid PROFILE on ALLOCATE command.
PROG_ERROR_NO_TRUNC	EIBERRCD = 08890000 (RECEIVE Only)
PROG_ERROR_PURGING	CDBERRCD = 08890000
RESOURCE_FAILURE_RETRY	EIBERRCD = A000
RESOURCE_FAILURE_NO_RETRY	EIBERRCD = A000
UNSUCCESSFUL	RETCODE = 01 04 04

APPC RETURN_CODE	CICS return codes
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.	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 deviates from the APPC architecture in a small number of detailed respects.

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 92 on page 371 illustrates the problems that can be caused by not syncpointing a sync level-2 conversation.

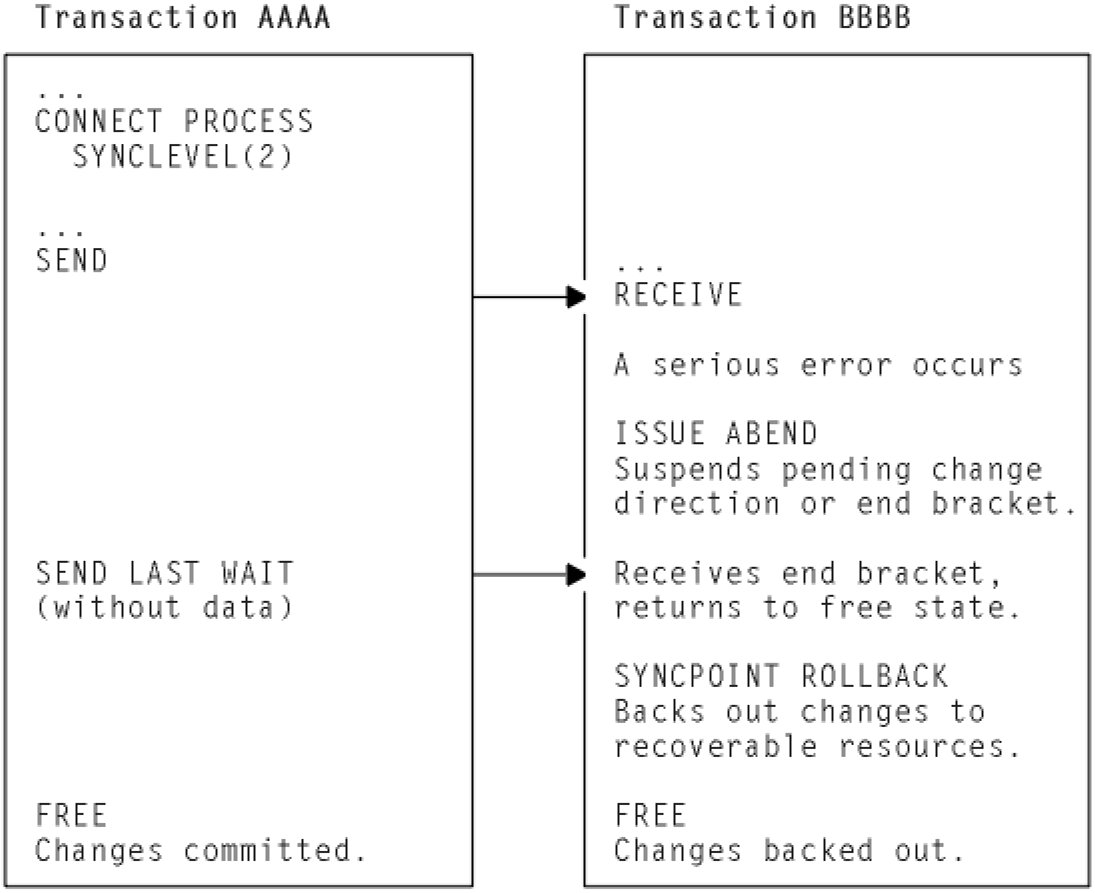


Figure 92. Losing data integrity on a sync level-2 conversation

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 *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. Use of the LOCKS(LONG) parameter results in less messages being passed on the APPC connection.
- 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 E. 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*.

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 90 on page 373 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 90 on page 373 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 90. 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” on page 374)	yes(“4” on page 374)	no

Table 90. Migration of LUTYPE6.1 programs to APPC links (continued)

Operation	Command	LU6.1	Migration	APPC
Start a partner transaction	SEND ATTACHID	yes("2" on page 374)	yes("5" on page 374)	no
Start a partner transaction	SEND FMH	yes("3" on page 374)	yes("6" on page 374)	no
Start a partner transaction	CONNECT PROCESS	no	yes("7" on page 374)	yes("7" on page 374)
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" on page 374)	yes
Synchronize	**Level 1** SEND CONFIRM ISSUE CONFIRMATION	no no	yes("8" on page 374) yes	yes yes
Synchronize	**Level 2** SEND CONFIRM ISSUE CONFIRMATION SYNCPOINT SYNCPOINT ROLLBACK	no no yes no	yes("8" on page 374) yes yes yes	yes yes yes yes

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 State transitions in APPC mapped conversations, 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 on the first column 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).
Abend *code*	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. When a different abend code applies, this is shown in the tables.
=	Remains in current state.
End	End of conversation.

State tables for LUTYPE6.1 migration-mode conversations

Tables showing the state transitions that occur when transactions engage in LUTYPE6.1 migration mode conversations, under the EXEC CICS API.

The ISSUE SIGNAL command and the EIBSIG flag

In the tables, the EIBSIG flag is not mentioned. 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.

The EIBSIG flag is set when the partner issues the **ISSUE SIGNAL** command.

The RECEIVE NOTRUNCATE command

The **RECEIVE NOTRUNCATE** command 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.

State changes for the SYNCPOINT and SYNCPOINT ROLLBACK commands

When the SYNCPOINT and SYNCPOINT ROLLBACK commands are issued, they are propagated on, and affect the state of, all the conversations that are currently active for the task, including MRO conversations.

Following rollback, the conversation can be in **SEND** or **RECEIVE** state, depending on the conversation state at the start of the current distributed unit of work. The conversation can be in **FREE** state if it ended abnormally due to session failure or due to deallocate abend being received, or if the partner transaction issued a SEND LAST WAIT or FREE command.

After a syncpoint or rollback, it is advisable to determine the conversation state before issuing any further commands against the conversation.

State changes following the ISSUE PREPARE command

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.

State tables

Table 91. States 1 - 6

Command issued	**EIB flag returned**	**Command returns**	**ALLO-CATED**	**SEND**	**PEND-RECEIVE**	**PEND-FREE**	**RECEIVE**	**CONF-RECEIVE**
			State 1	**State 2**	**State 3**	**State 4**	**State 5**	**State 6**
BUILD ATTACH	×	Immediately	=	=	=	=	=	=
EXTRACT ATTACH	×	Immediately	INVREQ	INVREQ	INVREQ	INVREQ	=	INVREQ
EXTRACT PROCESS (back-end transaction only)	×	Immediately	Abend	=	=	=	=	=
EXTRACT ATTRIBUTES	×	Immediately	=	=	=	=	=	=
SEND (any valid form)	EIBERR + EIBSYNRB	After error flow detected	Abend	13	13	13	Abend	Abend
SEND (any valid form)	EIBERR + EIBFREE	After error flow detected	12	12	12	12	Abend	Abend
SEND (any valid form)	EIBERR	After error flow detected	Abend	5	5	5	Abend	Abend
SEND INVITE WAIT	×	After data flows	5	5	Abend	Abend	Abend	Abend
SEND INVITE CONFIRM	×	After response from partner	5	5	Abend	Abend	Abend	Abend
SEND INVITE	×	After data buffered	3	3	Abend	Abend	Abend	Abend
SEND LAST WAIT	×	After data flows	12	12	Abend	Abend	Abend	Abend
SEND LAST CONFIRM	×	After response from partner	12	12	Abend	Abend	Abend	Abend
SEND LAST	×	After data buffered	4	4	Abend	Abend	Abend	Abend
SEND WAIT	×	After data flows	2	=	Abend	Abend	Abend	Abend
SEND CONFIRM	×	After response from partner	2	=	5	12	Abend	Abend
SEND	×	After data buffered	2	=	Abend	Abend	Abend	Abend
RECEIVE	EIBERR + EIBSYNRB	After rollback flow detected	Abend	13	13	Abend	13	Abend
RECEIVE	EIBERR + EIBFREE	After error detected	Abend	12	12	Abend	12	Abend
RECEIVE	EIBERR	After error detected	Abend	5	5	Abend	=	Abend
RECEIVE	EIBSYNC + EIBFREE	After sync flow detected	Abend	11	11	Abend	11	Abend
RECEIVE	EIBSYNC + EIBRECV	After sync flow detected	Abend	9	9	Abend	9	Abend

Table 91. States 1 - 6 (continued)

Command issued	**EIB flag returned**	**Command returns**	**ALLO-CATED**	**SEND**	**PEND-RECEIVE**	**PEND-FREE**	**RECEIVE**	**CONF-RECEIVE**
			State 1	**State 2**	**State 3**	**State 4**	**State 5**	**State 6**
RECEIVE	EIBSYNC	After sync flow detected	Abend	10	10	Abend	10	Abend
RECEIVE	EIBCONF + EIBFREE	After confirm flow detected	Abend	8	8	Abend	8	Abend
RECEIVE	EIBCONF + EIBRECV	After confirm flow detected	Abend	6	6	Abend	6	Abend
RECEIVE	EIBCONF	After confirm flow detected	Abend	7	7	Abend	7	Abend
RECEIVE	EIBFREE	After error flow detected	Abend	12	12	Abend	12	Abend
RECEIVE	EIBRECV	When data available	Abend	5	5	Abend	=	Abend
RECEIVE NOTRUNCATE	EIBCOMPL	When data available	Abend	5	5	Abend	=	Abend
RECEIVE	×	When data available	Abend	=	2	Abend	2	Abend
CONVERSE	As for RECEIVE		As for RECEIVE	As for RECEIVE	As for RECEIVE	As for RECEIVE	As for RECEIVE	As for RECEIVE
ISSUE CONFIRMATION	×	Immediately	Abend	Abend	Abend	Abend	Abend	5
ISSUE ERROR	EIBFREE	After response from partner	Abend	12	12	Abend	12	12
ISSUE ERROR	×	After response from partner	Abend	=	2	Abend	2	2
ISSUE ABEND	×	Immediately	Abend	12	12	12	12	12
ISSUE SIGNAL	×	Immediately	Abend	=	=	Abend	=	=
ISSUE PREPARE	EIBERR + EIBSYNRB	After response from partner	INVREQ	13	13	13	INVREQ	INVREQ
ISSUE PREPARE	EIBERR + EIBFREE	After error detected	INVREQ	12	12	12	INVREQ	INVREQ
ISSUE PREPARE	EIBERR	After error detected	INVREQ	5	5	5	INVREQ	INVREQ
ISSUE PREPARE	×	After response from partner	INVREQ	10	9	11	INVREQ	INVREQ
SYNCPOINT	EIBRLDBK	After response from partner	=	2 or 5	2 or 5	2 or 5	Abend ASP2	Abend ASP2
SYNCPOINT	×	After response from partner	=	=	5	12	Abend ASP2	Abend ASP2
SYNCPOINT ROLLBACK	×	After rollback across UOW	=	2 or 5	2 or 5	2 or 5	2 or 5	2 or 5
WAIT	×	Immediately	Abend	=	5	12	Abend	Abend
FREE	×	Immediately	End	End	Abend	End	Abend	Abend

Table 92. States 7 - 13

Command issued	**EIB flag returned**	**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**
BUILD ATTACH	×	=	=	=	=	=	=	=
EXTRACT ATTACH	×	INVREQ	INVREQ	INVREQ	INVREQ	INVREQ	INVREQ	INVREQ
EXTRACT PROCESS (back-end transaction only)	×	=	=	=	=	=	=	=
EXTRACT ATTRIBUTES	×	=	=	=	=	=	=	=

Table 92. States 7 - 13 (continued)

Command issued	**EIB flag returned**	**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**
SEND (any valid form)	EIBERR + EIBSYNRB	Abend	Abend	Abend	Abend	Abend	Abend	Abend
SEND (any valid form)	EIBERR + EIBFREE	Abend	Abend	Abend	Abend	Abend	Abend	Abend
SEND (any valid form)	EIBERR	Abend	Abend	Abend	Abend	Abend	Abend	Abend
SEND INVITE WAIT	×	Abend	Abend	Abend	Abend	Abend	Abend	Abend
SEND INVITE CONFIRM	×	Abend	Abend	Abend	Abend	Abend	Abend	Abend
SEND INVITE	×	Abend	Abend	Abend	Abend	Abend	Abend	Abend
SEND LAST WAIT	×	Abend	Abend	Abend	Abend	Abend	Abend	Abend
SEND LAST CONFIRM	×	Abend	Abend	Abend	Abend	Abend	Abend	Abend
SEND LAST	×	Abend	Abend	Abend	Abend	Abend	Abend	Abend
SEND WAIT	×	Abend	Abend	Abend	Abend	Abend	Abend	Abend
SEND CONFIRM	×	Abend	Abend	Abend	Abend	Abend	Abend	Abend
SEND	×	Abend	Abend	Abend	Abend	Abend	Abend	Abend
RECEIVE	EIBERR + EIBSYNRB	Abend	Abend	Abend	Abend	Abend	Abend	Abend
RECEIVE	EIBERR + EIBFREE	Abend	Abend	Abend	Abend	Abend	Abend	Abend
RECEIVE	EIBERR	Abend	Abend	Abend	Abend	Abend	Abend	Abend
RECEIVE	EIBSYNC + EIBFREE	Abend	Abend	Abend	Abend	Abend	Abend	Abend
RECEIVE	EIBSYNC + EIBRECV	Abend	Abend	Abend	Abend	Abend	Abend	Abend
RECEIVE	EIBSYNC	Abend	Abend	Abend	Abend	Abend	Abend	Abend
RECEIVE	EIBCONF + EIBFREE	Abend	Abend	Abend	Abend	Abend	Abend	Abend
RECEIVE	EIBCONF + EIBRECV	Abend	Abend	Abend	Abend	Abend	Abend	Abend
RECEIVE	EIBCONF	Abend	Abend	Abend	Abend	Abend	Abend	Abend
RECEIVE	EIBFREE	Abend	Abend	Abend	Abend	Abend	Abend	Abend
RECEIVE	EIBRECV	Abend	Abend	Abend	Abend	Abend	Abend	Abend
RECEIVE NOTRUNCATE	EIBCOMPL	Abend	Abend	Abend	Abend	Abend	Abend	Abend
RECEIVE	×	Abend	Abend	Abend	Abend	Abend	Abend	Abend
CONVERSE	As for RECEIVE	As for RECEIVE	As for RECEIVE	As for RECEIVE	As for RECEIVE	As for RECEIVE	As for RECEIVE	As for RECEIVE
ISSUE CONFIRMATION	×	2	12	Abend	Abend	Abend	Abend	Abend
ISSUE ERROR	EIBFREE	12	12	12	12	12	Abend	Abend
ISSUE ERROR	×	2	2	2	2	2	Abend	Abend
ISSUE ABEND	×	12	12	12	12	12	Abend	Abend
ISSUE SIGNAL	×	=	=	=	=	=	Abend	Abend
ISSUE PREPARE	EIBERR + EIBSYNRB	INVREQ	INVREQ	INVREQ	INVREQ	INVREQ	INVREQ	INVREQ
ISSUE PREPARE	EIBERR + EIBFREE	INVREQ	INVREQ	INVREQ	INVREQ	INVREQ	INVREQ	INVREQ
ISSUE PREPARE	EIBERR	INVREQ	INVREQ	INVREQ	INVREQ	INVREQ	INVREQ	INVREQ
ISSUE PREPARE	×	INVREQ	INVREQ	INVREQ	INVREQ	INVREQ	INVREQ	INVREQ
SYNCPOINT	EIBRLDBK	Abend	Abend	2 or 5	2 or 5	2 or 5	=	Abend
SYNCPOINT	×	Abend	Abend	2	2	12	=	Abend
SYNCPOINT ROLLBACK	×	2 or 5	2 or 5	2 or 5	2 or 5	2 or 5	=	2 or 5
WAIT	×	Abend	Abend	Abend	Abend	Abend	Abend	Abend
FREE	×	Abend	Abend	Abend	Abend	Abend	End	Abend

Appendix F. 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

Although you can use similar sequences of commands for a APPC mapped conversations and MRO conversations, there are some cases where the same command sequences operate differently in each conversation type.

Some of the differences between APPC mapped and MRO conversations are shown in the command sequence in Table 93 on page 379.

Table 93. 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 G. Below the SNA interface

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, and 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:

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, unless data is sent with the CONFIRM option.

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 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.

Notices

This information was developed for products and services offered in the United States of America. This material might be available from IBM in other languages. However, you may be required to own a copy of the product or product version in that language in order to access it.

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 rights 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 grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
United States of America

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

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 jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who want 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 Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119 Armonk,
NY 10504-1785
United States of America

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 Client Relationship Agreement, IBM International Programming License Agreement, or any equivalent agreement between us.

The performance data discussed herein is presented as derived under specific operating conditions. Actual results may vary.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to actual people or business enterprises is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Programming interface information

IBM CICS supplies some documentation that can be considered to be Programming Interfaces, and some documentation that cannot be considered to be a Programming Interface.

Programming Interfaces that allow the customer to write programs to obtain the services of CICS Transaction Server for z/OS, Version 5 Release 6 (CICS TS 5.6) are included in the following sections of the online product documentation:

- Developing applications
- Developing system programs
- CICS TS security
- Developing for external interfaces
- Application development reference
- Reference: system programming
- Reference: connectivity

Information that is NOT intended to be used as a Programming Interface of CICS TS 5.6, but that might be misconstrued as Programming Interfaces, is included in the following sections of the online product documentation:

- Troubleshooting and support
- CICS TS diagnostics reference

If you access the CICS documentation in manuals in PDF format, Programming Interfaces that allow the customer to write programs to obtain the services of CICS TS 5.6 are included in the following manuals:

- Application Programming Guide and Application Programming Reference
- Business Transaction Services

- Customization Guide
- C++ OO Class Libraries
- Debugging Tools Interfaces Reference
- Distributed Transaction Programming Guide
- External Interfaces Guide
- Front End Programming Interface Guide
- IMS Database Control Guide
- Installation Guide
- Security Guide
- CICS Transactions
- CICSPlex System Manager (CICSPlex SM) Managing Workloads
- CICSPlex SM Managing Resource Usage
- CICSPlex SM Application Programming Guide and Application Programming Reference
- Java Applications in CICS

If you access the CICS documentation in manuals in PDF format, information that is NOT intended to be used as a Programming Interface of CICS TS 5.6, but that might be misconstrued as Programming Interfaces, is included in the following manuals:

- Data Areas
- Diagnosis Reference
- Problem Determination Guide
- CICSPlex SM Problem Determination Guide

Trademarks

IBM, the IBM logo, and ibm.com® are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at Copyright and trademark information at www.ibm.com/legal/copytrade.shtml.

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Apache, Apache Axis2, Apache Maven, Apache Ivy, the Apache Software Foundation (ASF) logo, and the ASF feather logo are trademarks of Apache Software Foundation.

Gradle and the Gradlephant logo are registered trademark of Gradle, Inc. and its subsidiaries in the United States and/or other countries.

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

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

The registered trademark Linux® is used pursuant to a sublicense from the Linux Foundation, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis.

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

Red Hat®, and Hibernate® are trademarks or registered trademarks of Red Hat, Inc. or its subsidiaries in the United States and other countries.

Spring Boot is a trademark of Pivotal Software, Inc. in the United States and other countries.

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

Zowe™, the Zowe logo and the Open Mainframe Project™ are trademarks of The Linux Foundation.

The Stack Exchange name and logos are trademarks of Stack Exchange Inc.

Terms and conditions for product documentation

Permissions for the use of these publications are granted subject to the following terms and conditions.

Applicability

These terms and conditions are in addition to any terms of use for the IBM website.

Personal use

You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of IBM.

Commercial use

You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Rights

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

IBM online privacy statement

IBM Software products, including software as a service solutions, (*Software Offerings*) may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user or for other purposes. In many cases no personally identifiable information (PII) is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect PII. If this Software Offering uses cookies to collect PII, specific information about this offering's use of cookies is set forth below:

For the CICSPlex SM Web User Interface (main interface):

Depending upon the configurations deployed, this Software Offering may use session and persistent cookies that collect each user's user name and other PII for purposes of session management, authentication, enhanced user usability, or other usage tracking or functional purposes. These cookies cannot be disabled.

For the CICSPlex SM Web User Interface (data interface):

Depending upon the configurations deployed, this Software Offering may use session cookies that collect each user's user name and other PII for purposes of session management, authentication, or other usage tracking or functional purposes. These cookies cannot be disabled.

For the CICSPlex SM Web User Interface ("hello world" page):

Depending upon the configurations deployed, this Software Offering may use session cookies that do not collect PII. These cookies cannot be disabled.

For CICS Explorer:

Depending upon the configurations deployed, this Software Offering may use session and persistent preferences that collect each user’s user name and password, for purposes of session management, authentication, and single sign-on configuration. These preferences cannot be disabled, although storing a user's password on disk in encrypted form can only be enabled by the user's explicit action to check a check box during sign-on.

If the configurations deployed for this Software Offering provide you, as customer, the ability to collect PII from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see IBM Privacy Policy and IBM Online Privacy Statement, the section entitled *Cookies, Web Beacons and Other Technologies* and the IBM Software Products and Software-as-a-Service Privacy Statement.

Index