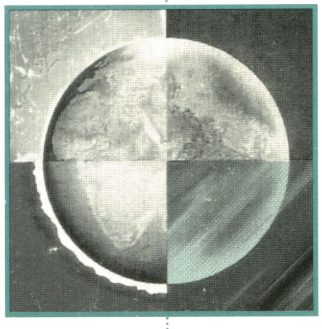
CICS Programmer's Toolkit

Programmer's Reference



INTERLINK
Computer Sciences

CICS Programmer's Toolkit

Programmer's Reference

Version 2.0

August 1994

Document Number 200801-024040-200100

Interlink Computer Sciences, Inc.
47370 Fremont Boulevard
Fremont, California 94538
U.S.A.
(510) 657-9800

IMPORTANT NOTICE

Second Edition (August 1994)

The information in this document applies to the Interlink CICS Programmer's Toolkit™ Version 2.0 software. If you would like additional copies of this or any Interlink documentation, contact the Customer Support Department at our corporate address.

Copyright

Disclaimer

All information in this document is subject to periodic change and revision.

While every effort has been made to ensure the information presented in this document is accurate, Interlink disclaims liability for any inaccuracies or omissions that may have occurred.

If you find information in this document that is incorrect, misleading, or incomplete, we would appreciate your comments and suggestions. Please use the reader comment form at the back of this manual to let us know how we may improve the information we provide you.

Trademarks

This Interlink document often refers to numerous hardware and software products by their trade names. In most, if not all cases, these designations are claimed as trademarks or registered trademarks by their respective companies.

Contents

200801-024040-200100

20 024040-200100

200801-024040-200100

200801-024040-200100

Chapter Four COBOL SUBROUTINE CALLS

200801-024040-200100

200801-024040-200100

Preface

Read this guide for detailed information about the CPT services and support for Assembler, COBOL, C, and PL/1 high-level languages. Read the ***CICS Programmer's Toolkit Installation and Administration Guide*** for information about installation and administration of the CPT Programmer's Toolkit.

200801-024040-200100

Audience

This guide is intended for these audiences:

- IBM CICS developers responsible for TCP/IP communication applications.

Organization

This guide includes these chapters and appendixes:

TITLE	DESCRIPTION
Chapter One – CPT API SERVICES	Describes the eight subroutines calls available in the CICS Programmer's Toolkit.
Chapter Two – ASSEMBLER CALLS	Provides detailed coding information for CPT assembler language subroutine calls. It includes information about conventions and terminology used to describe the associated data structures, and defines the basic forms and formats that apply.
Chapter Three – C SUBROUTINE CALLS	Provides detailed coding information for CPT C language function calls. It includes information about conventions and terminology used to describe the associated data structures, and defines the basic forms and formats that apply.
Chapter Four– COB0L SUBROUTINE CALLS	Provides detailed coding information for CPT COBOL language subroutine calls. It includes information about conventions and terminology used to describe the associated data structures, and defines the basic forms and formats that apply.
Chapter Five – PL/I SUBROUTINE CALLS	Provides detailed coding information for CPT PL/I language subroutine calls. It includes information about conventions and terminology used to describe the associated data structures, and defines the basic forms and formats that apply.

Additional Information

Refer to the these documents for additional information:

- From Interlink Computer Sciences
  - ***SNS/API Programmer's Reference Manual***
- From IBM
  - ***CICS/MVS Version 2.1.2 (or higher) Application Programmer's Reference Manual***
  - ***CICS/MVS Version 2.1.2 (or higher) Resource Definition Macro Manual***
  - ***CICS/MVS Version 2.1.2 (or higher) Resource Definition Online Manual***

Command Notation

The command notation used in this guide follows specific rules. This table illustrates these conventions:

CONVENTION	DESCRIPTION	EXAMPLE
Initial Capitalized	Window and pop-up titles.	Use the Local Output Data Set Specifications window to specify the MVS data set.
Bold, Initial Capitalized	Menu titles, menu options, functions, and fields.	Enter your name in the **User Name** field, your password in the **Password** field, and then click on **Accept.** **Note:** Click always refers to buttons.
BOLD, ALL CAPITALS	Keyboard keys.	Press **RETURN**.
Italic	The first use of words, acronyms, and phrases included in the glossary, if a glossary is included.	The address parameter is called the *mask*.
Nine point Courier Bold Underline	Default values in JCL statements, console commands, and Assembler operations.	These values may be set: `[SYNC | ASYNC][,LOCAL | REMOTE]`
`Nine point Courier`	System output, data entry, example code, and file names. **Note:** As data entry, the text appears exactly as you should enter it, including capitalization.	◆ The message `SHOW MESSAGES` displays. ◆ The `lpd` server receives the data. ◆ Set the value of this argument to `TRUE`
`Nine point Courier italic`	Data that you must specify or supply input for, or variable information. **Note:** This convention may appear in conjunction with nine point Courier bold. or nine point Courier	◆ This is the command syntax `df` *`x nnn hostname`* ◆ The system displays this message: `Process started for ID` *`num`*
`[ ]` (Square brackets)	Optional arguments or commands.	`PORT = ({ipaddress | hostname}[,port])` `LPDEFAULT=(':key=value[:key=value...]')`
`|` (Vertical bar)	A logical *or* indicating that you can select or specify one or the other of the specified values.	
`{ }` (Curly braces)	Indicates a list of possible items. Generally, one item appears in the message.	
`...` (Ellipsis)	Indicates that an item may repeat multiple times.	

200801-024040-200100

Chapter 1
CPT API SERVICES

This chapter provides information about the CPT API services. It includes these sections:

- Overview

  Provides a brief overview of this implementation of the CPT API facility and its subroutine calls.
- Connection Management

  Describes how to use the `LISTEN` and `CONNECT` services to provide connection management.
- Data Transfer

  Describes how to use the `SEND` and `RECEIVE` services to provide data transfer.
- Connection Release

  Describes how to use the `CLOSE` service to release a connection.
- Data Translation

  Describes how to use the `TRANSLATE` service to provide single-byte character set translation.
- Facility Management

  Describes how to use the `GIVE` and `TAKE` services to provide facility management.
- Sample CPT API Pseudo Code

  Provides sample pseudo codes for client and server applications.
- CPT API Sample Programs

  Provides a table listing each sample program and its corresponding language, and sample client and server programs that are in the `T09SAMP` data set.

200801-024040-200100

Overview

Implementation of the CPT API services is controlled through various subroutine calls. There are internal subroutine calls used to support the CPT environment and external subroutine calls used by applications for service requests. The internal calls are used to manage resources associated with connections and the *Task-Related User Exit* (*TRUE*) interface. The external calls are used to generate service requests related to specific application tasks.

The CPT environment management programs are responsible for initialization, logging, and termination of the TRUE interface. The application management programs are responsible for functions directly associated with user-written applications. The application management routines are primarily concerned with the recovery and cleanup of CICS, and non-CICS resources associated with user-written applications during task termination.

There are some pseudo code samples illustrating the use of the CPT API services at the end of this chapter.

CPT Task-Related User Exit Interface (TRUE)

CPT uses the CICS general-use programming interface facility called task-related user exit (TRUE). The TRUE interface allows applications access to an external, or non-CICS, resource. The external CICS resource utilized by CPT is a communication subsystem based on open network protocols. The communication subsystem is an *Application Program Interface* (*API*) to a transport provider.

Application Programming Concepts

The CPT API facility supports communication with open network protocols using a client/server model. The CPT API services are designed to communicate with the transport layer of the Basic Reference Model of *Open System Interconnection* (*OSI*).

A server application passively listens, or waits, for a connection request. Once a connection indication from a client application has been received and established, data transfer can begin. The server specifies a transport provider address or port where it listens for connection requests. This port is called a *well-known port.*

The client application actively connects to a server application. The client contacts a well-known port for a server. A client determines the server's host and port where it initiates the connection. If the server is not listening, the connection request fails. Once a connection is established, data transfer begins.

Both a client and server application can transfer data simultaneously over a full duplex connection. Any dependence on data flow control is application specific.

200801-024040-200100

TCP Connection Management

TCP connection management is accomplished using the LISTEN and CONNECT services. These services are responsible for the creation of resources and for the establishment of connections. A connection is represented by a token.

The token is returned to the application in the Argument for Connection Management (ACM). The token is used for all subsequent CPT service requests related to that connection. Multiple connections or tokens can be obtained by an application. However, the mechanism used to manage the connections is controlled by the application.

TCP connection management services associate ownership of a newly established connection to the calling task. This provides the TRUE management routines the ability to release resources during normal or abnormal task termination. Ownership of resources can be controlled automatically by internal CPT routines, or explicitly by an application through facility management services.

TCP connection management services set the operating environment for the connection. Optional arguments specify transport provider buffering, CPT internal tracing, connection statistics, and subtask initialization. Such information can only be specified by connection management services and cannot be modified after a connection has been established.

Information related to the newly established connection is returned within the ACM. This information contains IP host names, IP addresses, transport provider addresses, and more. The information can be used by the application or ignored.

LISTEN

The LISTEN service is used by a user-written application to passively listen for connection requests. This ability provides the application with server support. The LISTEN service requires an ACM to be initialized by the user application and a call to the LISTEN service routine.

Successful completion of the LISTEN service returns a token that represents the established connection with a client. This token is used for all data transfer, data processing, and connection termination service requests.

Two variations of the LISTEN service allow a data processing transaction to be initiated internally. The data processing transaction can be predetermined by specifying the transid in the connection management argument or dynamically by the connecting client. This option is selected by initializing a field within the connection management argument. Completion of the LISTEN service is generally indicated by an error at CPT or transport provider termination.

CONNECT

The CONNECT service is used by a user-written application to actively establish a connection with a server, thus providing it with client support. The CONNECT service requires an ACM to be initialized by the user application and requires a call to be made to the CONNECT service routine.

Successful completion of the CONNECT service returns a token that represents the established connection with a server. This token is used for all subsequent data transfer, data processing, and connection termination service requests.

200801-024040-200100

TCP Data Transfer

TCP data transfer is accomplished using the SEND and RECEIVE services. These services are responsible for reliable transmission of data to and from the transport provider's API. Data Transfer services require an established connection and a user application buffer.

The transport provider is not responsible for record or file boundaries. It cannot be assumed that data transmitted will be received with the same logical boundaries with which it was sent. Record and file boundaries are transparent to the transport provider. Thus, applications should be designed with some mechanism to distinguish logical record or file boundaries.

File boundaries may be the easiest to distinguish. It is possible that a connection release could indicate the designated end of file, that the sender has completed transmitting all data, and is closing its half of the full duplex connection. The receiver can transmit data or simply close the connection.

If record orientated data is to be transmitted, then some pre-determined mechanism used by both the client and server applications should be designed. Mechanisms such as separator character(s), fixed length records, or record header information can be used to delimit records. These mechanisms are also used by the CPT tools.

The TCP data transfer services have several options that make programming for stream oriented data easier. There are two variations of a timed RECEIVE call that specify the amount of data to receive before returning to the caller. There is an option to send and/or receive data in logical records where the length of the record is stored in the first two bytes of the record. There is also an option to send and/or receive data in logical records where the records are separated by a predefined character sequence.

SEND

The SEND service is used by a user-written application to send or output data over the connection. The SEND service requires an Argument for Data Transfer (ADT) to be initialized by the application and requires a call to be issued to the SEND service. The data transfer argument contains a token, data buffer address, and data buffer length.

Upon completion, a return code field in the ADT indicates success or failure of request.

RECEIVE

The RECEIVE service is used by a user-written application to receive or input data from the connection. The RECEIVE service requires an ADT to be initialized by the application and requires a call to be issued to the RECEIVE service. The data transfer argument contains a token, data buffer address and data buffer length.

Upon completion, a return code field in the ADT indicates success or failure of the request. The data transfer length field must be retrieved to determine the amount of data received.

UDP Data Transfer and Endpoint Creation

Data transfer for UDP is accomplished using the SENDTO and RCVFROM services. These services also create an endpoint if the caller does not pass an existing endpoint in the argument for data transfer. UDP endpoints are represented by a token.

UDP does not provide the reliable data transmission capabilities that TCP does. UDP works as well as the underlying IP internet and hardware network. Applications developed for local area networks are probably quite reliable while the same applications ported to a wide area internet might not be. UDP applications generally should be developed with logic to account for datagrams that are lost or out of sequence.

Because reliability is not built into connectionless data transmission, there is no corresponding overhead for the transport provider. This makes UDP data transmission faster than TCP data transmission. Since there is no notion of a connection between two UDP endpoints, whenever data is sent or received it is transmitted all at once. Applications do not have to be designed to extract logical records from variable length streams of data.

SENDTO

The SENDTO service is used by a user-written application to send a datagram to a remote UDP endpoint. The SENDTO service requires an argument for data transfer (ADT) to be initialized by the application, which must include a buffer address, buffer length, and remote endpoint address identification. If an existing token is not passed, new token, send, and receive buffer queues are created. The size and number of CPT SEND and RECEIVE buffers for the endpoint can be set in the ADT along with optional trace and statistics flags.

RCVFROM

The RCVFROM service is used by a user-written application to receive datagrams from remote UDP endpoints. The RCVFROM service requires an ADT to be initialized by the application, which must include a buffer address and buffer length. If an existing token is not passed, new token, send, and receive buffer queues are created. When a new token is to be created, the local well-known UDP port must also be passed in the ADT. The size and number of CPT SEND and RECEIVE buffers for the endpoint can be set in the ADT along with optional trace and statistics flags.

200801-024040-200100

Connection and Endpoint Release

Connection and endpoint release is accomplished using the CLOSE service. This service is responsible for the release of the connection and all internal CPT associated resources. Connection Release requires either a listen or data transfer connection to be established.

A connection or endpoint release is scheduled explicitly by issuing the CLOSE service request, or implicitly by the TRUE management routines during task termination. If an explicit CLOSE service is issued and no connections or endpoints are owned by the task, the implicit close scheduled by the TRUE management routines will not be issued.

TRUE management routines are responsible for managing connections and associated resources. The releasing of resources is one facility provided by the task-related user task management routines and is controlled by an ownership mechanism. During task termination the TRUE management routines automatically (implicitly) schedule a connection or endpoint release (CLOSE) request for owned resources. CLOSE, issued by the TRUE management routines for active connections, is abortive.

The facility management services can be used to manipulate connections, endpoints, and associated resources owned by a task to avoiding implicit termination.

CLOSE

A user-written application uses the CLOSE service to release the connection or endpoint. The CLOSE service requires an Argument for Close (ACL) to be initialized by the application and requires a call to be issued to the CLOSE service. The ACL contains a token and termination options. The termination options include orderly (graceful) and abortive connection release.

Upon completion, a return code field in the ACL indicates success or failure of the request. When a connection or endpoint has been successfully released, the token is no longer valid.

Data Translation

The TRANSLATE service provides support for single-byte character set translation. This implies that any character set of 256 (or less) data representations is supported. Translation service requires an established connection and a user application buffer.

Applications with special translation requirements are able to select an alternate translation table. Alternate translation tables must be customized to the CPT system by applying an SMP/E usermod. Read **Chapter Three – INSTALLATION AND CONFIGURATION** in the ***CICS Programmer's Toolkit Installation and Administration Guide*** for a detailed description of Translation Table Customization.

TRANSLATE

The TRANSLATE service uses a user-written application to translate EBCDIC and ASCII data within a user buffer. The TRANSLATE service requires an Argument for Translation (AXL) to be initialized by the application and requires a call to be issued to the TRANSLATE service. The AXL contains a token, data buffer address and length, and translation options. Translation options indicate EBCDIC to ASCII or ASCII to EBCDIC translation. Optionally, a user application can override the site default translation table.

Upon completion, a return code field in the AXL indicates success or failure of the request.

200801-024040-200100

Facility Management

The GIVE and TAKE services provide facility management. These optional services provide enhanced connection management support for multi-tasked applications. Facility management services require an established connection. A CPT connection that is used by several CICS tasks can define a multi-task application. For example, the LISTEN and RECEIVE tools used in conjunction create a multi-task application.

A multi-threaded server application is an example of a multi-tasked application where the CPT connection is established by a listening task and then a data processing transaction is initiated to handle data transfer. Any application that is designed to have multiple tasks processed by a single CPT connection can benefit from facility management services.

A client or single-threaded server application that establishes a connection, transfers data, and releases the connection all within the same task, does not need to use the facility management services.

CPT connection management services (LISTEN and CONNECT) create connections. By default, the task that issues a connection management service obtains ownership of the connection and its associated resources. CPT TRUE management routines are responsible for managing connections and their associated resources. Releasing resources is one facility provided by the TRUE management routines and is controlled by an ownership mechanism. During task termination the TRUE management routines automatically (implicitly) schedule a connection release (CLOSE) request for owned resources.

The release of a connection and its associated resources is performed through the explicit connection release request, or the implicit task termination release facility. The GIVE and TAKE services affect the implicit task termination release facility by disabling (GIVE) and enabling (TAKE) ownership of a connection.

There is no restriction on the number of times a multi-tasked application can issue a GIVE or TAKE facility management service. The mechanism used to pass information related to a CPT connection between tasks is application-dependent.

024040-200100

GIVE

A user-written application uses the GIVE service to disable ownership of internal CPT resources associated with a connection. This facility prohibits CPT task-related user task management routines from releasing a connection and associated resources during task termination. The GIVE service requires an Argument for Facility Management (AFM) to be initialized by the application and requires a call to be issued to the GIVE service. The version number and token are the only arguments required.

The GIVE service provides a mechanism to disable the TRUE task termination routine from releasing the connection and associated resources, thereby allowing a connection and its associated resources to remain available after task termination. This facility enhances multi-tasked application design.

Connections, and their associated resources, that have been given must be taken by other tasks or explicitly released. Otherwise, the connections and

resources persist indefinitely. Resources that are not taken can lead to hung connections, storage shortages within the CICS region or the transport provider, or unpredictable results.

A connection can be closed via the CLOSE service after it has been given. The GIVE service only affects implicit release management services provided by the CPT task-related user task management routines. Also, a connection that can be taken is not required to be given. There is no restriction that a connection and its associated resources must be given before they can be taken.

Upon completion, a return code field in the AFM indicates success or failure of the request.

TAKE

A user-written application uses the TAKE service to obtain ownership of internal CPT resources associated with a connection. This facility enables CPT TRUE management routines to release a connection and its associated resources during task termination. The TAKE service requires that an AFM be initialized by the application and requires a call to be issued to the TAKE service. The version number and token are the only arguments required.

A connection that will be taken is not required to be given. There is no restriction that a connection and its associated resources is given before it can be taken. This provides a mechanism for ensuring proper connection and resource termination, while still allowing a connection to be used by several tasks.

The TAKE service is implemented implicitly within the SEND, RECEIVE, and TRANSLATE services. This implies that the connection is automatically associated with the last task that issued a SEND, RECEIVE, or TRANSLATE service request. Therefore, if a connection has been previously given by the current task, an additional GIVE service request is required to release ownership of the connection.

The implicit TAKE service within the SEND, RECEIVE, and TRANSLATE services allow facility management to be handled by the CPT TRUE management routines. Hence, the TAKE and, to some extent, GIVE facility management services are optional.

Upon completion, a return code field in the AFM indicates success or failure of the request.

Read **Chapter Two – ASSEMBLER CALLS, Chapter Three – C SUBROUTINE CALLS, Chapter Four – COBOL SUBROUTINE CALLS**, and **Chapter Five – PL/1 SUBROUTINE CALLS** for detailed information about the subroutine calls for each language. For installation and configuration information, read **Chapter Three – INSTALLATION AND CONFIGURATION** in the ***CICS Programmer's Toolkit Installation and Administration Guide***.

200801-024040-200100

Security Exit

CPT provides a Security Exit for user evaluation of requests for the services of local listeners/servers. If a Security Exit is implemented, the user program is invoked for each connection request in a TCP environment. The appropriate server transaction is initiated if authorized by the user security program. Otherwise, the client is notified that the connection is terminated.

To implement the Security Exit in CPT, the `SCTYEXIT=program-name` must be coded in the `T09MCICS` macro of the `T09CONFG` Configuration Table. The user program will be CICS LINKed during the connection process and must conform to CICS coding standards and be defined as a Processing Program Table (PPT) entry.

- If no `SCTYEXIT=program-name` is coded in the Configuration Table, all connection requests will be authorized.
- If `SCTYEXIT=program-name` is coded but the program is missing or is disabled, no connections will be permitted.

Note: The second condition may be checked by invoking the Administrator Interface panel for the Configuration Table; however, the Security Exit program is displayed only if it is disabled or if it is not in the PPT.

The Security Communications Block

The connection process and the user security program communicate through the Security Communications Block. Connection provides information about the request and its origin. The user program will determine whether the request is to be authorized and, optionally, the name of a terminal facility to be associated with a STARTed server transaction. A DSECT of the Security Communications Block for Assembler programs may be generated with the `T09DSCTY` macro.

FIELD	FORMAT	DESCRIPTION
SECTRAN	4-byte character	Requested server transaction
SECDATA	40-byte character	Client data, if available
SECSTRT	2-byte character	Method of server initiation: KC, TC, or IC
SECICTM	6-byte character	IC Hours, Minutes, Seconds
SECAFAM	halfword binary	Address family: Inet domain=2
SECRPRT	halfword binary	Client remote port number
SECRHST	fullword binary	Client remote host IP address
SECACTN	1-byte character	Authorization switch: ◆ Character '1' = authorized ◆ Other = not authorized
	1-byte character	Reserved filler

20 024040-200100

FIELD	FORMAT	DESCRIPTION
SECTMID	4-byte character	Associated terminal facility.
SECLPRT	halfword binary	Requested server local port
SECLHST	fullword binary	Local host IP address

The Security Exit Program

The user application program is responsible not only for making the determination of whether a connection is authorized, by also for any desired logging or other capture of unauthorized requests. Because the exit will be driven for each connection request, performance implications should be considered in designing the user program.

The only fields which are checked upon return from the user program are the authorization switch and the associated terminal facility. If the transaction is authorized by virtue of a character '1' in the action field, then the server transaction will be initiated. If the terminal facility has been changed from any CICS termid associated with the listener to a new CICS termid, then the authorized transaction will be STARTed with that new termid, and any applicable CICS security associated with the new termid will prevail.

The security exit is invoked only for connections for which the CPT Listen service will start the server transaction. This may be by way of a transaction specified in ACMTRNID or in a tool, or by use of the Client-Data feature. In other client/server designs, the application receives control when the connection is made, and the application should make any desired security checks before beginning server activity.

200801-024040-200100

Sample CPT API Pseudo Code

This section provides some examples of pseudo code for client and server applications

Client Application Example

A CICS program is required to send and receive data to a server application residing on a workstation. The CICS application reads and writes to temporary storage. The CICS application is required to initiate the connection and send the first packet.

The workstation or server's IP host name is SATURN and the well-known port address on that machine is 1234. The server's data representation is ASCII. The server application expects data from the client and responds with data.

The CICS client application attempts to establish a connection with the server before processing any data. The client application reads temporary storage, then translates the data into ASCII before sending it to the server. The client application is then required to receive a response from the server. The data received must be translated into EBCDIC before it can be written to temporary storage. The application loops until all data has been processed, then closes the connection gracefully. Any unexpected error causes the connection to terminate abnormally.

```
.
. Working Storage
.
    Define Storage for Connection Management Argument
    Define Storage for Data Transfer Argument
    Define Storage for Data Translation Argument
    Define Storage for Connection Release Argument

.
. Initialize Connection Management Argument and issue CONNECT service.
.
    Set transport protocol to connection-mode (TCP).
    Set server well-known port to 1234.
    Set server IP host name to 'SATURN'.
    Call CONNECT service with Connection Management Argument.
    Check CONNECT service Return Code.
        If Return Code not zero, then log error and GOTO RETURN.

.
. Retrieve connection Token.
.
    Copy TOKEN from Connection Management Argument.

.
. Read Temporary Storage Queue and check for end of queue.
.
READ_NEXT_TS label:
    EXEC CICS READQ TS QUEUE(tsqname1) SET( ) LENGTH( )
        If Handle Condition is QEMPTY, then GOTO CLOSE_ORDERLY.
        If Handle Condition error, then GOTO CLOSE_ABORTIVE.
```

20-024040-200100

```
.
. Initialize Data Translation Argument and issue TRANSLATE service.
.
   Set connection TOKEN.
   Set translation from EBCDIC to ASCII.
   Set address of translation data buffer.
   Set length of translation data buffer.
   Call TRANSLATE service with Data Translation Argument.
   Check TRANSLATE service Return Code.
      If Return Code error, then GOTO CLOSE_ABORTIVE.

.
. Initialize Send Data Transfer Argument and issue SEND service.
.
   Set connection TOKEN.
   Set address of send data buffer.
   Set length of send data buffer.
   Call SEND service with Data Transfer Argument.
   Check SEND service Return Code.
      If Return Code error, then GOTO CLOSE_ABORTIVE.

.
. Initialize Receive Data Transfer Argument and issue RECEIVE service.
.
   Set connection TOKEN.
   Set address of receive data buffer.
   Set length of received data buffer.
   Call RECEIVE service with Data Transfer Argument.
   Check RECEIVE service Return Code.
      If Return Code error, then GOTO CLOSE_ABORTIVE.

.
. Retrieve length of network data RECEIVE service processed.
.
   Copy RECEIVE service data length from Data Transfer Argument.

.
. Initialize Data Translation Argument and issue TRANSLATE service.
.
   Set connection TOKEN.
   Set translation from ASCII to EBCDIC.
   Set address of translation data buffer.
   Set length of translation data buffer.
   Call TRANSLATE service with Data Translation Argument.
   Check TRANSLATE service Return Code.
      If Return Code error, then GOTO CLOSE_ABORTIVE.

.
. Write Data to Temporary Storage Queue.
.
   EXEC CICS WRITEQ TS QUEUE(tsqname2) SET( ) LENGTH( )
      If Handle Condition error, then GOTO CLOSE_ABORTIVE.

.
. Loop application for more temporary storage data.
.
   GOTO READ_NEXT_TS.

.
. Initialize Connection Release Argument and issue CLOSE service.
.
```

200801-024040-200100

```
CLOSE_ORDERLY label:
    Set connection TOKEN.
    Set orderly release option.
    Call CLOSE service with Connection Release Argument.
    Check CLOSE service Return Code.
       If Return Code error, then log error.
    GOTO RETURN.

.
. Initialize Connection Release Argument and issue CLOSE service.
.
CLOSE_ABORTIVE label:
    Set connection TOKEN.
    Set abortive release option.
    Call CLOSE service with Connection Release Argument.
    Check CLOSE service Return Code.
       If Return Code error, then log error.

.
. Terminate Task
.
RETURN label:
    EXEC CICS RETURN
```

024040-200100

Server Application Example 1

A CICS program is required to receive and send data from a client application. The CICS server application listens for connection indications and then echoes any received data back to the client. Termination of the server application is determined by a CICS or API (transport provider) shutdown condition.

The CICS server application listens for connection indications on well-known port 2000. This server application handles data transfer in-stream and does not initiate additional client connections until the current connection has been terminated. Therefore, this is a single-threaded server application. The application loops within the CPT receive/send logic until a CPT release indication is determined and then closes the connection gracefully. Any unexpected error while receiving and sending data causes the connection to be terminated abnormally.

The `LISTEN` service request returns two tokens. One token represents the data transfer connection and the other token represents the server connection. The data transfer token is used with send and receive processing, while the listen token can only be used during task termination.

```
.
. Working Storage
.
    Define Storage for Connection Management Argument
    Define Storage for Data Transfer Argument
    Define Storage for Connection Release Argument

.
. Initialize Connection Management Argument and issue LISTEN service.
.
    Set transport protocol to connection-mode (TCP).
    Set server well-known port to 2000.

LISTEN_LOOP label:
    Call LISTEN service with Connection Management Argument.
    Check LISTEN service Return Code.
    If Return Code equal CICS shutdown, then GOTO CLOSE_LISTEN.
        If Return Code equal API shutdown, then GOTO CLOSE_LISTEN.
        If Return Code unknown, then log error and GOTO
CLOSE_LISTEN.

.
. Retrieve Data Transfer Connection and Listen Tokens.
.
    Copy DT_TOKEN from Connection Management Argument.
    Copy LISTEN_TOKEN from Connection Management Argument.
.
. Initialize Receive Data Transfer Argument and issue RECEIVE service.
.

ECHO_LOOP label:
    Set connection DT_TOKEN.
    Set address of receive data buffer.
    Set length of received data buffer.
    Call RECEIVE service with Data Transfer Argument.
    Check RECEIVE service Return Code.
        If Return Code equal RELEASE, then GOTO CLOSE_ORDERLY.
        If Return Code error, then GOTO CLOSE_ABORTIVE.
```

200801-024040-200100

```
.
. Retrieve length of network data RECEIVE service processed.
.
    Copy RECEIVE service data length from Data Transfer Argument.

.
. Initialize Send Data Transfer Argument and issue SEND service.
.
    Set connection DT_TOKEN.
    Set address of send data buffer.
    Set length of send data buffer.
    Call SEND service with Data Transfer Argument.
           If Return Code error, then GOTO CLOSE_ABORTIVE.

.
. Loop application for more client data.
.
    GOTO ECHO_LOOP.

.
. Initialize Connection Release Argument and issue CLOSE service.
.
CLOSE_ORDERLY label:
    Set connection DT_TOKEN.
    Set orderly release option.
    Call CLOSE service with Connection Release Argument.
    Check CLOSE service Return Code.
       If Return Code error, then log error.
    GOTO SERVER_LOOP.

.
. Initialize Connection Release Argument and issue CLOSE service.
.
CLOSE_ABORTIVE label:
    Set connection DT_TOKEN.
    Set abortive release option.
    Call CLOSE service with Connection Release Argument.
    Check CLOSE service Return Code.
       If Return Code error, then log error.
    GOTO SERVER_LOOP.

.
. Initialize Connection Release Argument and issue CLOSE service.
.
CLOSE_LISTEN label:
    Check for LISTEN_TOKEN.
       If no LISTEN_TOKEN, then GOTO RETURN.
    Set connection LISTEN_TOKEN.
    Set orderly release option.
    Call CLOSE service with Connection Release Argument.
    Check CLOSE service Return Code.
              If Return Code error, then log error.

.
. Terminate Task
.
RETURN label:
    EXEC CICS RETURN
```

20-024040-200100

Server Application Example 2

This example illustrates a multi-threaded CICS server application. In this example, the CICS server application listens for connection indications and starts a data processing transaction. Termination of the server application is determined by a CICS or API (transport provider) shutdown condition.

The CICS server application listens for connection indications on well-known port 3000. Once a connection has been established, the connection management GIVE service is issued to release ownership of the connection. A CICS START command is then issued for a data processing transaction. Any unexpected error causes the data transfer connection to terminate abnormally.

The LISTEN service request returns two tokens, one token represents the data transfer connection and the other represents the server connection. The data transfer token is passed to the data processing transaction, while the listen token can only be used during task termination.

```
.
. Working Storage
.
    Define Storage for Connection Management Argument
    Define Storage for Facility Management Argument
    Define Storage for Connection Release Argument

.
. Initialize Connection Management Argument and issue LISTEN
service.
.
    Clear Server Listen Token DT_TOKEN
    Set transport protocol to connection-mode (TCP).
    Set server well-known port to 3000.

LISTEN_LOOP label:
    Call LISTEN service with Connection Management Argument.
    Check LISTEN service Return Code.
        If Return Code equal CICS shutdown, then GOTO
CLOSE_LISTEN.
        If Return Code equal API shutdown, then GOTO CLOSE_LISTEN.
        If Return Code unknown, then log error and GOTO
CLOSE_LISTEN.

.
. Retrieve Data Transfer Connection and Listen Tokens.
.
    Copy DT_TOKEN from Connection Management Argument.
    Copy LISTEN_TOKEN from Connection Management Argument.

.
. Initialize Facility Management Argument and issue GIVE service.
.
    Set connection DT_TOKEN.
    Call GIVE service with Facility Management Argument.
    Check GIVE service Return Code.
        If Return Code error, then log error GOTO CLOSE_ABORTIVE.

.
. Start Data Transfer Transaction.
.
```

200801-024040-200100

```
    EXEC CICS START TRANSID(transid) FROM(DT_TOKEN) LENGTH(4)
       If Handle Condition error, then GOTO CLOSE_ABORTIVE.

.
. Loop for additional connection indications.
.
    GOTO LISTEN_LOOP.

.
. Initialize Connection Release Argument and issue CLOSE service.
.
CLOSE_ABORTIVE label:
    Set connection DT_TOKEN.
    Set abortive release option.
    Call CLOSE service with Connection Release Argument.
          Check CLOSE service Return Code.
       If Return Code error, then log error.
    GOTO LISTEN_LOOP.

.
. Initialize Connection Release Argument and issue CLOSE service.
.
CLOSE_LISTEN label:
    Check for LISTEN_TOKEN.
       If no LISTEN_TOKEN, then GOTO RETURN.
    Set connection LISTEN_TOKEN.
    Set orderly release option.
    Call CLOSE service with Connection Release Argument.
    Check CLOSE service Return Code.
       If Return Code error, then log error.

.
. Terminate Task
.
RETURN label:
    EXEC CICS RETURN
```

Server Application Example 3

This example illustrates a data processing program associated with a multi-threaded server application. The transaction is initiated by a server program after a client connection has been established. The program is responsible for processing data associated with a connection.

The TAKE service is an optional facility and is provided implicitly through the SEND, RECEIVE, and TRANSLATE services.

The application loops within the CPT receive/send logic until a CPT release indication is determined, then closes the connection gracefully. Any unexpected error while receiving and sending data causes the connection to terminate abnormally.

```
.
. Working Storage
.
   Define Storage for Facility Management Argument
   Define Storage for Data Transfer Argument
   Define Storage for Data Translation Argument
   Define Storage for Connection Release Argument

.
. Obtain Data Transfer Token for Server Transaction.
.
   EXEC CICS RETRIEVE FROM(TOKEN) LENGTH(4)
      If Handle Condition error, then GOTO CLOSE_ABORTIVE.

.
. Initialize Facility Management Argument and issue TAKE service.
.
   Set connection TOKEN.
   Call TAKE service with Facility Management Argument.
   Check TAKE service Return Code.
      If Return Code error, then log error GOTO CLOSE_ABORTIVE.

.
. Initialize Receive Data Transfer Argument and issue RECEIVE service.
.
RECV_LOOP label:
   Set connection TOKEN.
   Set address of receive data buffer.
   Set length of received data buffer.
   Call RECEIVE service with Data Transfer Argument.
   Check RECEIVE service Return Code.
      If Return Code equal RELEASE, then GOTO CLOSE_ORDERLY.
      If Return Code error, then GOTO CLOSE_ABORTIVE.

.
. Retrieve length of network data RECEIVE service processed.
.
   Copy RECEIVE service data length from Data Transfer Argument.

.
. Initialize Data Translation Argument and issue TRANSLATE service.
.
   Set connection TOKEN.
   Set translation from ASCII to EBCDIC.
   Set address of translation data buffer.
   Set length of translation data buffer.
```

200801-024040-200100

```
    Call TRANSLATE service with Data Translation Argument.
    Check TRANSLATE service Return Code.
       If Return Code error, then GOTO CLOSE_ABORTIVE.

.
. Application to process input and determine output.
.

.
. Initialize Data Translation Argument and issue TRANSLATE service.
.
    Set connection TOKEN.
    Set translation from EBCDIC to ASCII.
    Set address of translation data buffer.
    Set length of translation data buffer.
    Call TRANSLATE service with Data Translation Argument.
    Check TRANSLATE service Return Code.
       If Return Code error, then GOTO CLOSE_ABORTIVE.

.
. Initialize Send Data Transfer Argument and issue SEND service.
.
    Set connection TOKEN.
    Set address of send data buffer.
    Set length of send data buffer.
    Call SEND service with Data Transfer Argument.
          Check SEND service Return Code.
       If Return Code error, then GOTO CLOSE_ABORTIVE.

.
. Loop application for more client data.
.
    GOTO RECV_LOOP.

.
. Initialize Connection Release Argument and issue CLOSE service.
.
CLOSE_ORDERLY label:
    Set connection TOKEN.
    Set orderly release option.
    Call CLOSE service with Connection Release Argument.
    Check CLOSE service Return Code.
       If Return Code error, then log error.
    GOTO RETURN.

.
. Initialize Connection Release Argument and issue CLOSE service.
.
CLOSE_ABORTIVE label:
    Set connection TOKEN.
    Set abortive release option.
    Call CLOSE service with Connection Release Argument.
    Check CLOSE service Return Code.
       If Return Code error, then log error.

.
. Terminate Task
.
RETURN label:
    EXEC CICS RETURN
```

20-024040-200100

Server Application Example 4

This example is a variation of the multi-threaded CICS server application shown in Server Application 2. The CICS server application listens for connection indications and then causes the LISTEN service to initiate a data transfer transaction. The initiated data transfer transaction could be Server Application 3. Termination of the server application is determined by a CICS or API (transport provider) shutdown condition.

The CICS server application listens for connection indications on well-known port 4000. A transaction ID is specified for the data transfer program. Once a connection is established, the connection management GIVE service and the CICS START command are issued from within the LISTEN service.

Return from the LISTEN service request does not occur until an error has occurred. The error could be either CPT and CICS termination, or some unexpected error. CPT or CICS termination is considered graceful termination, while anything else produces an error.

```
.
. Working Storage
.
    Define Storage for Connection Management Argument
    Define Storage for Connection Release Argument

.
. Initialize Connection Management Argument and issue LISTEN
service.
.
    Set transport protocol to connection-mode (TCP).
    Set server well-known port to 4000.
    Set Data Transactions ID.

    Call LISTEN service with Connection Management Argument.
    Check LISTEN service Return Code.
        If Return Code equal CICS shutdown, then  GOTO
CLOSE_LISTEN.
        If Return Code equal API shutdown, then GOTO CLOSE_LISTEN.

.
. Log LISTEN Service Unknown error
.
    Log Connection Management Return Code.

.
. Initialize Connection Release Argument and issue CLOSE service.
.
CLOSE_LISTEN label:
    Copy DT_TOKEN from Connection Management Argument.
    Check for LISTEN_TOKEN.
        If no LISTEN_TOKEN, then GOTO RETURN.
    Set connection LISTEN_TOKEN.
    Set orderly release option.
    Call CLOSE service with Connection Release Argument.
    Check CLOSE service Return Code.
        If Return Code error, then log error.

.
. Terminate Task
.
RETURN label:
    EXEC CICS RETURN
```

200801-024040-200100

CPT API Sample Programs

These sample programs are in the CPTSAMP data set that was unloaded when the CICS Programmer's Toolkit was installed. Descriptions of each program follows where *x* denotes the programming language from the table shown here.

This table illustrates the sample program name and its corresponding language:

CPTSAMP MEMBER NAME	LANGUAGE	TYPE
T09PACL1	Assembler	TCP Client 1 program
T09PACL2	Assembler	TCP Client 2 program
T09PASV1	Assembler	TCP Server 1 program
T09PASV2	Assembler	TCP Server 2 program
T09PASV3	Assembler	TCP Server 3 program
T09PASV4	Assembler	TCP Server 4 program
T09PACLU	Assembler	UDP Client program
T09PASVU	Assembler	UDP Server Program
T09PCCL1	COBOL	TCP Client 1 program
T09CCL2	COBOL	TCP Client 2 program
T09PCSV1	COBOL	TCP Server 1 program
T09PCSV2	COBOL	TCP Server 2 program
T09PCSV3	COBOL	TCP Server 3 program
T09PCSV4	COBOL	TCP Server 4 program
T09PCCLU	COBOL	UDP Client program
T09PCSVU	COBOL	UDP Server program
T09PPCL1	PL/I	TCP Client 1 program
T09PPCL2	PL/1	TCP Client 2 program
T09PPSV1	PL/I	TCP Server 1 program
T09PPSV2	PL/I	TCP Server 2 program
T09PPSV3	PL/I	TCP Server 3 program
T09PPSV4	PL/I	TCP Server 4 program
T09PPCLU	PL/1	UDP Client program
T09PPSVU	PL/1	UDP Server program
T09PSCL1	C	TCP Client 1 program
T09PSCL2	C	TCP Client 2 program

024040-200100

CPTSAMP MEMBER NAME	LANGUAGE	TYPE
T09PSSV1	C	TCP Server 1 program
T09PSSV2	C	TCP Server 2 program
T09PSSV3	C	TCP Server 3 program
T09PSSV4	C	TCP Server 4 program
T09PSCLU	C	UDP Client program
T09PSSVU	C	UDP Server program

Client 1 Sample Program

T09PxCL1 is an example of a client program that sends a message, (input at a terminal) to a server program. It uses an LL (length) convention to indicate when all data has been sent. It sends the length first followed by the message. The server (s) echoes back the LL and data. When the message is fully received, the client requests an orderly close of the connection.

This program is initiated at a terminal by entering the transaction ID, a server transaction ID, and a text variable. If a server transaction ID omitted, the echo port is requested. If a text variable is omitted, a dummy message is substituted.

TCP Client 2 Sample Program

T09PxCL2 is an example of a client program that sends a message to a server program and then receives it back. The Client 2 sample uses special processing options that cause CPT to format the stream data into logical records. These SEND and RECEIVE options make logical record programming much easier from the CPT application standpoint.

These are the logical record options:

- Logical record based on separator characters
- Logical record based on length set in the first two data bytes
- The receiver defines what a full record length is and waits until it receives that amount

This program is initiated at a terminal by typing in the transaction ID followed by an option: FULL (default), LL, or SEP. T09PxCL2 sends the data to the TCP Echo server.

200801-024040-200100

TCP Server 1 Sample Program

T09PxSV1 is an example of a server program, that can be initiated either during CICS start up or dynamically using a supplied transaction ID. The server issues a listen on a specific port and then remains active in CICS as a long-running task. When a client program designates the same port for a connect, CPT initiates this server for receive-and-send handshaking.

In this example, the server echoes back messages received from the client. After the client requests an orderly release from the connection, the server goes back to passive listening on the port. This server is single-threaded; any subsequent requests for its services wait until preceding clients have completed and closed connections.

TCP Server 2 Sample Program

T09PxSV2 is an example of a server program that does not issue a listen, but takes the connection from the original listener. It is initiated by CPT when a listening task detects a client request for the port number assigned to this server. CPTPXSV2 can be initiated directly by another transaction that is a listening server, by CPT from a listening transaction's specification of ACMTRNID in its connection management argument, or by a listener specified in a T09MLSTN statement in the CPT tool configuration table.

In this example, the server receives one or more messages from the client, then echoes it back. When the client requests a release, or when an error occurs, the server disconnects and goes away.

A fresh copy of the server is activated as needed.

Server 3 Sample Program

T09PxSV3 is an example of a server program that can be initiated either during CICS start up or dynamically using a supplied transaction ID. The server issues a listen on a specific port and continues to remain active in the system as a long running task. When a client transaction requests the service associated with its port, T09PxSV3 is activated to connect with that client.

In this example, when the server is awakened to service a client, it spawns another task to do the complex work requested by the client. This frees the long-running server up to initiate a new listen and to respond to additional clients in a timely manner.

This server task terminates when CPT is stopped.

20-024040-200100

Server 4 Sample Program

T09PxSV4 is an example of a server program that can be initiated either during CICS start up or dynamically using a supplied transaction ID. The server issues a listen for a specific service, but also provides CPT with a transaction name for an independent task to be started when a client requests a connection to the service. That task does any complex work associated with the service, while the server continues as a long-running task that listens for additional requests for the service

This server task terminates when CPT is stopped.

UDP Client Sample Program

T09PxCLU is an example of a UDP client program that calls the SENDTO service to send a datagram, input at a terminal, to a server program that echoes the datagram back. The default server is the UDP echo server with T09PXSVU being the other possible destination by specifying the associated transaction ID. When the datagram has been received back via the RCVFROM service, the sample client closes the endpoint.

UDP Server Sample Program

T09PxSVU is an example of a UDP server program that hangs a RCVFROM on a well-known port and waits for incoming datagrams. When RCVFROM completes, the server calls the SENDTO service to send the datagram back to its originator.

This program should be initiated as a started transaction.

200801-024040-200100

Chapter 2
ASSEMBLER CALLS

This chapter describes the assembler subroutine calls of CICS/API. These are:

- CLOSE
- CONNECT
- GIVE
- LISTEN
- RCVFROM
- RECEIVE
- SEND
- SENDTO
- TAKE
- TRANSLATE

It also describes the macro instruction T09MCALL, return codes of the T09DRTCD macro instruction, and the CICS storage requirements for a TCP connection or a UDP endpoint.

All messages are in **Appendix C – MESSAGES AND CODES** in the ***CICS Programmer's Toolkit Installation and Administration Guide***.

200R01-024040-200100

CLOSE

The CLOSE service closes an established connection. Both orderly (or graceful) and abortive termination options are supported. The CLOSE service performs all associated functions required for CPT resource clean-up.

To invoke the CLOSE service, a user application is required to first build an Argument for Close (ACL) and then to issue a call to the CLOSE routine. Valid arguments include the ACL version number, connection token, and termination options. On completion, a return code is set to indicate success or failure of the request.

This table describes the arguments for the CLOSE service:

MACRO ID	DSECT NAME	SIZE	CREATED BY
T09DACL	ACL	30 (X'1E')	User application

Assembler Data Area

This is what the DSECT control block looks like in Assembler language:

```
Name      Operation    Operands     Description

ACL       DSECT        ,
ACLVERS   DS           H          Version number
ACLFUNC   DS           H          Function code
ACLTOKEN  DS           A          Token (CEP)
          DS           A          Reserved
          DS           F          Reserved
ACLRTNCD  DS           F          Return code
ACLDGNCD  DS           F          Diagnostic code
ACLOPCDS  DS           0F         Termination Option Codes
ACLOPCD4  DS           X          Termination Option Code 4
ACLOPCD3  DS           X          Termination Option Code 3
ACLOPCD2  DS           X          Termination Option Code 2
ACLOPCD1  DS           X          Termination Option Code 1
ACLORDER  EQU          X'00'      - Orderly Release
ACLABORT  EQU          X'01'      - Abortive Release
          DS           H          Reserved
```

NAME	OPERATION
ACLVERS	Version number Indicates the CPT version number of the argument list used by the calling program. This required field must be set to a binary 2 for this release of CPT. Default: None
ACLFUNC	Function code Indicates the function or callable service ID requested by the application program. This field is not set by the application, but is initialized by the *Task-Related User Exit* (*TRUE*) interface stub program. Default: None

200801-024040-200100

NAME	OPERATION
ACLTOKEN	Connection or endpoint token Specifies a token that represents a TCP connection, a TCP listening end point, or a UDP end point. A token is created by the TCP connection initiation routines or by the UDP data transfer and endpoint creation routines. **Note:** The token is required. Default: None
ACLRTNCD	Return code Indicates the return code set by the CLOSE service. This value is also returned in register 15 and indicates the success or failure of the service. Default: 0
ACLDGNCD	Diagnostic code Indicates the diagnostic code set by the service request. This value generally indicates a transport provider return code. Default: 0
ACLOPCDS	Specifies CLOSE service processing control options. These are the supported options: ◆ ACLORDER – Indicates a graceful termination. This option implements orderly release of the TCP/IP connection. This is the preferred option for terminating a connection, and is used when processing has completed successfully. ◆ ACLABORT – Indicates abortive termination. This option implements a disconnect or reset of the TCP/IP connection. This option is generally used after an unrecoverable application error has occurred. **Note:** The notion of orderly or abortive CLOSE for a UDP endpoint is meaningless and the options specified when calling CLOSE for a UDP token are not important. CPT knows if the token is UDP and will close it properly. Default: ACLORDER

Completion Information

The CLOSE service completes normally when the connection is terminated and associated resources are released. Graceful termination waits for all pending transport provider asynchronous SEND and RECEIVE requests to complete. Graceful termination also waits for both ends of the full-duplex connection to close. Abortive termination closes the transport provider connection without regard to pending transport provider requests. Abortive termination cause data loss and should be used only when data integrity is not required.

On normal return to the application program, the general return code in register 15 (ACLRTNCD) is set to zero (CPTIRCOK). The diagnostic code in register 0 (ACLDGNCD) is always zero.

If the CLOSE service completes abnormally, some user data may be lost. The general return code (ACLRTNCD) in register 15, and the diagnostic code (ACLDGNCD) in register 0, indicate the nature of the failure. The diagnostic code (ACLDGNCD) may contain a specific code which identifies a particular transport provider error.

20-024040-200100

Return Codes

The CLOSE service returns a code in register 15 and 0 that indicates the results of the execution. These values are in the ACLRTNCD (R15) and ACLDGNCD (R0) within the ACL. The diagnostic code is optional and indicates the transport provider return code. Read **Appendix C – MESSAGES AND CODES** in the ***CICS Programmer's Toolkit Installation and Administration Guide*** for the return code cross reference table.

This table describes the CLOSE service return codes:

RETURN	DIAGNOSTIC CODE	DESCRIPTION
CPTIRCOK		Successful
CPTEVERS		Control block version number is not supported.
CPTETOKN		Specified token is not valid.
CPTEPRGE	Yes	CPT Interface is terminating.
CPTENAPI	Yes	Transport provider API is not available.
CPTETERM	Yes	Environment is being terminated.
CPTERLSE	Yes	Release indication.
CPTEDISC	Yes	Disconnect indication.
CPTEINTG	Yes	Transport provider API integrity error.
CPTEENVR	Yes	Transport provider API environment error.
CPTEFRMT	Yes	Transport provider API format error.
CPTEPROC	Yes	Transport provider API procedure error.
CPTABEND		Abnormal exception occurred.
CPTEOTHR	Yes	An undefined exception occurred.

200801-024040-200100

Usage Information

The CLOSE service terminates an established transport provider endpoint and releases associated resources. Established transport provider endpoints can be half of a TCP connection, a TCP listening endpoint, or a UDP endpoint, and are represented by a token.

The CLOSE service utilizes the ACL. The CLOSE service requires the application to set the ACL version number and token fields. Optional control information related to termination processing can be specified. The address of the ACL is required to be loaded into register 1 before the CLOSE service.

The version number (ACLVERS) indicates the CPT release level in which this user application program is written. This required field must be set to a binary 1 and is validated by the CLOSE service before it processes the request.

The function code (ACLFUNC) indicates the CPT callable service ID. The field is not initialized by a user application program and has little value to the application except for dump analysis. The function code can identify and map an argument list with the error or trace log and dump analysis.

The token (ACLTOKEN) indicates the connection and internal resources that are to be released. This is a required field and is validated by the CLOSE service before processing the request.

The ACLOPCDS field specifies CLOSE processing control options and provides a mechanism for event notification on return to the application program. Currently, ACLORDER and ACLABORT are the only two options supported; no facility exists for CLOSE event notification, except by way of return code values.

If the option code ACLORDER is selected, the CLOSE service completes all pending transport provider requests. These requests represent previous asynchronous SENDs and/or RECEIVEs that have neither completed yet, nor had their completion checked. This may require the CLOSE service to block the application. This option will then perform an orderly release of the TCP/IP connection. This is the preferred mechanism for connection termination.

If the option code ACLABORT is selected, the CLOSE service terminates the connection and no attempt is made to preserve data in transit. The remote user will receive a disconnect indication.

20-024040-200100

Example:

This example establishes a connection, processes data, and closes the connection. The token is loaded from the Argument for Connection Management (ACM) and used by all of the following CPT service requests. The ACL version number and the token are set before the CLOSE service is issued. No termination option is specified, so orderly release is selected as the default. Register 15 is checked (on return from the CLOSE service) and, if successful, no error is logged:

```
*
*   Dsect's
*
    T09DACM MF=DSECT            Argument for Connection Management
    T09DACL MF=DSECT            Argument for Connection Release
*
*   Working storage
*
DFHEISTG  DSECT
          .
          .
          .
ACMARG    DS     XL(ACMLEN)    Argument for Connection Management
CLOSEARG  DS     XL(ACLLEN)    Argument for Connection Release
*
*         Entry
*
label     DFHEIENT
          .
          . CPT Connection Management initialization and request
          .
          L      R9,ACMTOKEN   Load ACM Token
          .
          . Application and CPT Data Transfer (SEND/RECEIVE) processing
          .
*
*   CPT Connection Termination
*
CLOSE     DS     0H
          LA     R7,CLOSEARG       Load CLOSE argument addr
          USING  T09DACL,R7
          MVC    ACLVERS,=H'2'     Set Version number
          ST     R9,ACLTOKEN       Save connection token
          LR     R1,R7             Load ADT address in Reg 1
          L      R15,=V(T09FCLOS) Load CLOSE service Stub address
          BALR   R14,R15           Issue CLOSE service
          LTR    R15,R15           Test Return Code
          BZ     END               Good, Terminate transaction
*
*         Connection Release Error
*
          L   R3,ACLRTNCDLoad ACL Return Code
          L   R4,ACLDGNCDLoad ACL Diagnostic Code
          .
          . Process and log application errors
          .
          DROP   R7                        ACL
*
*         Terminate Transaction
*
END       DS     0H
          EXEC   CICS RETURN
```

200801-024040-200100

Example:

In this example, an established connection receives an error while processing data and aborts the connection. The ACL version and token are specified. The ACL abort option is selected to indicate the type of connection termination required. Register 15 is checked (on return from the CLOSE service) to determine request completion status.

```
*
*       Dsect's
*
        T09DACL MF=DSECT        Argument for Connection Release
*
*       Working storage
*
DFHEISTG  DSECT
          .
          .
          .
CLOSEARG  DS     XL(ACLLEN)    Argument for Connection Release
*
*         Entry
*
label     DFHEIENT
          .
          . CPT Connection Management request
          .
          L   R9,ACMTOKEN          Load ACM Token
          .
          .CPT Data Transfer (SEND/RECEIVE) processing
          .
          LTR    R15,R15           Test service return code
          BNZ    DTERROR           Non-zero, Abort connection
          .
          . Application processing
          .
          B      LOOP              Data processing loop
*
*         Abort connection
*
DTERROR   DS     0H
          LA     R5,CLOSEARG       Load CLOSE argument address
          USING  T09DACL,R5
          MVC    ACLVERS,=H'2'     Set Version number
          ST     R9,ACLTOKEN       Save connection token
          MVI    ACLOPCD1,ACLABORTSet Abortive
          LR     R1,R5             Load ADT address in Reg 1
          L      R15,=V(T09FCLOS) Load CLOSE service Stub address
          LTR    R15,R15           Test Return Code
          BZ     END               Good, Terminate transaction
*
*       Connection Release Error
*
          L      R3,ACLRTNCD       Load ACL Return Code
          L      R4,ACLDGNCD       Load ACL Diagnostic Code
          .
          . Process and log application errors
          .
          DROP   R5                ACL
*
*         Terminate Transaction
*
END DS  0H
        EXEC   CICS RETURN
```

Example: This example terminates a single-thread server application. A server application can contain two CPT connections; the first for the data transfer connection and the second for the server or listening connection. The tokens are loaded from the ACM. A `CLOSE` service is issued for both connections. This example uses the `T09MCALL` macro instruction to set the version number and to issue the `CLOSE` service call. Register `15` is checked (on return from each service call) and, if successful, processing continues.

```
*
*          Dsect's
*
           T09DACM MF=DSECT     Argument for Connection Management
           T09DACL MF=DSECT     Argument for Connection Release
*
*          Working storage
*
DFHEISTG   DSECT
           .
           .
           .
LSTNARG    DS     XL(ACMLEN)    Argument for Connection Management
CLOSEARG   DS     XL(ACLLEN)    Argument for Connection Release
*
*          Entry
*
label      DFHEIENT
           .
           . CPT Server Connection Management Request
           .
           LTR    R15,R15       Test Return Code
           BNZ    ERROR         Non-zero, Log Error
           L      R9,ACMTOKEN   Load Data Transfer Token
           L      R10,ACMTLSTN  Load Listening Token
           DROP   R2            ACM
           .
           . Application and CPT data transfer (SEND/RECEIVE)
              processing
           .
*
*          CPT Connection Termination
*
CLOSEDT    DS     0H
           LA     R7,CLOSEARG      Load CLOSE argument addr
           USING  T09DACL,R7
           ST     R9,ACLTOKEN      Save Data Transfer token
           MVI    ACLOPCD1,ACLORDER   Set Graceful
           T09MCALL CLOSE,PARM=CLOSEARG  Issue Close request
           LTR    R15,R15          Test Return Code
           BZ     CLOSESRV         Good, Close Server Connection
*          Data transfer Connection Release Error
           L      R3,ACLRTNCD      Load ACL Return Code
           L      R4,ACLDGNCD      Load ACL Diagnostic Code
           .
           . Process and log application errors
           .
CLOSESRV   DS     0H
           XC     ACL(ACLLEN),ACL  Clear CLOSE argument
           ST     R10,ACLTOKEN     Save Listen token
           MVI    ACLOPCD1,ACLORDER   Set Graceful
           T09MCALL CLOSE,PARM=CLOSEARG  Issue Close request
           LTR    R15,R15          Test Return Code
           BZ     END              Good, Terminate transaction
*          Server (Listening) Connection Release Error
           L      R3,ACLRTNCD      Load ACL Return Code
           L      R4,ACLDGNCD      Load ACL Diagnostic Code
           .
           . Process and log application errors
           .
```

200801-024040-200100

```
                DROP   R7                    ACL
*
*               Terminate Transaction
*
END             DS     0H
                EXEC   CICS RETURN
```

CONNECT

This service provides a client facility for use by an application program. The CONNECT service establishes a session with the local transport provider, then actively connects to a server. When connection is established with a server, the CONNECT service returns control to the calling program. Information related to the connection is updated and returned within the ACM.

To invoke the CONNECT service, a user application is required to first build an ACM and then to issue a call to the CONNECT routine. The minimum information required by this service is version number, server host address, and well-known port. Optional information related to data transfer buffering, CPT statistics and tracing, and subtask initialization can be specified.

This table describes the CONNECT service arguments:

MACRO ID	DSECT NAME	SIZE	CREATED BY
T09DACM	ACM	676 (X'2A4')	User application

Assembler Data Area

This is what the DSECT control block looks like in Assembler language:

```
Name      Operation Length    Description

ACM       DSECT     ,
ACMVERS   DS        H         Version number
ACMFUNC   DS        H         Function code
ACMTOKEN  DS        A         Token (CEP)
          DS        A         Reserved
          DS        F         Reserved
ACMRTNCD  DS        F         Return code
ACMDGNCD  DS        F         Diagnostic code
ACMSTATS  DS        0F        Statistics flag
          DS        XL3
ACMSTAT   DS        X         Primary statistics request byte
ACMSCONN  EQU       X'01'      - Connection statistics
ACMSTERM  EQU       X'02'      - Termination statistics

ACMTRACE  DS        0F        Trace flag
          DS        XL2
ACMTRAC2  DS        X         Second trace byte (for high level lang)
ACMTTKNS  EQU       X'01'      - Trace token information
ACMTTPL   EQU       X'02'      - Trace TPL block
ACMTRLSE  EQU       X'04'      - Trace release information
ACMTSTOR  EQU       X'08'      - Trace getmain/freemain
ACMTCLTD  EQU       X'10'     - Trace TD from client (IBM style)
ACMTRAC1  DS        X         First trace byte (for high level lang)
ACMTNTRY  EQU       X'01'      - Trace entry points
ACMTARGS  EQU       X'02'      - Trace arguments
ACMTRECV  EQU       X'04'      - Trace trecv
ACMTSEND  EQU       X'08'      - Trace tsend
ACMTTERM  EQU       X'10'      - Trace termination
ACMTPASS  EQU       X'20'      - Trace take
ACMTCLSE  EQU       X'40'      - Trace close
ACMTTERR  EQU       X'80'      - Trace TPL errors
```

200801-024040-200100

```
ACMQSEND   DS      F       tsend queue size
ACMMSEND   DS      F       Maximum tsend TPL buffer size
ACMQRECV   DS      F       trecv queue size
ACMMRECV   DS      F       Maximum trecv TPL buffer size
ACMTLSTN   DS      F       Listen token
ACMUCNTX   DS      F       User context field
ACMTRNID   DS      CL4     Transaction ID
           DS      X       Reserved for C string
           DS      XL3     Unused - available
ACMLPORT   DS      H       Local transport provider port
ACMRPORT   DS      H       Remote transport provider port
ACMSRVCE   DS      CL36    Local/remote service name
           DS      X       Reserved for C string
           DS      X       Unused - available
ACMOPTNS   DS      0H      ACM option flags
ACMOPTN2   DS      X       ACM option flag 2
ACMOPTN1   EQU     X       ACM option flag 1
ACMNODNR   EQU     X'04'    - No local/remote name resolution
ACMLTRAN   EQU     X'02'    - Listen start transaction
*                             specified in first 1-4 bytes of
*                             client data
ACMSYNC    EQU     X'01'    - Issue syncpoint for listen
ACMLADDR   DS      A       Local IP host address
ACMRADDR   DS      A       Remote IP host address
ACMLNAME   DS      CL255   Local IP host name
           DS      X       Reserved for C string
ACMRNAME   DS      CL255   Remote IP host name
           DS      X       Reserved
           DS      H       Reserved
           DS      H       Reserved
ACMTIMEO   DS      F       Timeout to recv client data
           DS      F
```

PARAMETER	DESCRIPTION
ACMVERS	Version. Indicates the version number of the CPT argument used by the calling program. This required field must be set to a binary 2 for this release of CPT. Default: None
ACMFUNC	Function code. Indicates the function or callable service ID requested by the application program. This field should not be set by the application, but rather is initialized by the TRUE interface stub program. Default: None (generated by service stub)
ACMTOKEN	TCP connection token. Specifies the token is created and returned by the CONNECT service. This token is used for all subsequent service calls for the client connection. Default: 0 (token returned)
ACMRTNCD	Return code. Indicates the return code set by the CONNECT service. This value is also returned in register 15 and indicates the success or failure of the service. Default: 0

20-024040-200100

PARAMETER	DESCRIPTION
ACMDGNCD	Diagnostic code. Indicates the diagnostic code received by the CONNECT service for a transport provider request. A detailed explanation of this value can be found in the transport provider's ***API Programmer's Reference Guide***. Default: 0
ACMSTATS	ACMSCONN | ACMSTERM Specifies statistics logging options for the application program. The facility can be used for debugging and tuning during development. ◆ ACMSCONN – Specifies that a message(s) be generated on establishing either a listen service or a data transfer connection. These messages are generated by the CPT LISTEN and CONNECT services. The message numbers associated with this option are CPT802I and CPT803I. ◆ ACMSTERM – Specifies that a message(s) is to be generated on terminating an established connection. These messages are generated by the CPT CLOSE service. The message numbers associated with this option are CPT807I, CPT808I, and CPT809I. Default: 0 (no statistics logging)
ACMTRACE	Specifies trace logging options for the application program. The facility can be used for debugging during development. ◆ ACMTNTRY – Specifies that a message be generated on entry to a CPT service routine. The message is generated by all CPT service routines. The message number associated with this option is CPT901I. ◆ ACMTARGS – Specifies that a hex dump of the caller's arguments be generated on entry and exit of a CPT service routine. Messages are generated by the corresponding service routine. Message numbers associated with this option are CPT902I, CPT903I, CPT904I, CPT911I, CPT912, CPT921I, CPT922I, CPT924I, CPT925I, CPT930I, and CPT932I. ◆ ACMTRECV – Specifies that a hex dump of the transport provider's input (RECEIVE) data be logged. Messages are generated by the CPT RECV service. The message number associated with this option is CPT913I. ◆ ACMTSEND – Specifies that a hex dump of the transport provider's output (SEND) data be logged. Messages are generated by the CPT SEND service. The message number associated with this option is CPT914I. ◆ ACMTTERM – Specifies that a message be generated on termination of a CPT service routine. Messages are generated by all CPT service routines. The message number associated with this option is CPT908I. ◆ ACMTPASS – Specifies that a hex dump of resources related to a passed connection be logged. Messages are generated by the CPT GIVE and TAKE services. The message numbers associated with this option are CPT919I and CPT920I. ◆ ACMTCLSE – Specifies that a hex dump of resources related to a CLOSE processing of a connection be logged. Messages are generated by the CPT CLOSE service. The message number associated with this option is CPT906I ◆ ACMTTERR – Specifies that a hex dump of a transport provider API parameter list that fails successful completion be logged. The message number associated with this option is CPT400I. ◆ ACMTTKNS – Specifies that a hex dump of the connection token be logged. Messages are generated by all services routines on entry. The message number associated with this option is CPT909I. ◆ ACMTTPL – Specifies that a hex dump of the transport provider API parameter list be logged. The message numbers associated with this option are CPT917I and CPT935I. ◆ ACMTRLSE – Specifies that a message indicating a transport provider release indication be logged. Messages are generated by various CPT service routines. The message number associated with this option is CPT414I. ◆ ACMTSTOR – Specifies that a hex dump of the storage management argument to be logged. Messages are generated by various CPT service routines. The message numbers associated with this option are CPT928I and CPT929I. ◆ ACMTCLTD – Trace transient data writes from the LISTEN service (used with the ACMLTRAN listen start transaction option. The message number associated with this option is CPT918I. Default: 0 (no trace logging)

200801-024040-200100

PARAMETER	DESCRIPTION
ACMQSEND	API send queue size. Specifies the maximum number of uncompleted SEND requests that can be queued by the application to the transport provider (API). This value lets applications control output processing and can affect throughput rates. The value is negotiated with and can be modified by the transport provider. The total allocation for output processing is the product of the SEND queue and buffer size values and cannot exceed 61K. Default: 4
ACMMSEND	API send buffer size. Specifies the maximum number of user data bytes that can be transferred by the application in a single SEND request to the transport provider (API). This value lets applications control output processing and can affect throughput rates. The value is negotiated with and can be modified by the transport provider. The total allocation for output processing is the product of the SEND queue and buffer size values cannot exceed 61K. Default: 4096
ACMQRECV	API receive queue size. Specifies the maximum number of uncompleted RECEIVE requests that can be queued by the application to the transport provider (API). This value lets applications control input processing and can affect throughput rates. The value is negotiated with and can be modified by the transport provider. The total allocation for input processing is the product of the RECEIVE queue and buffer size values cannot exceed 61K. Default: 4
ACMMRECV	API receive buffer size. Specifies the maximum number of user data bytes that can be transferred by the application, in a single RECEIVE request, to the transport provider (API). This value lets applications control input processing and can affect throughput rates. The value is negotiated with and can be modified by the transport provider. The total allocation for input processing is the product of the RECEIVE queue and buffer size values cannot exceed 61K. Default: 1024
ACMTLSTN	Listen service token. This field is not used by the CONNECT service. The value in this field is not validated nor is it modified. Default: None
ACMUCNTX	One word of user context. Specifies one arbitrary word of user context to be associated with the connection. The information provided is not interpreted by CPT, and is merely saved with other connection information. Default: 0 (no user context)
ACMTRNID	Listen start transaction ID. This field is not used by the CONNECT service. The value in this field is not validated nor is it modified. Default: None
ACMLPORT	Listen well-known service port. This value represents the TCP port on the local host that was assigned to the client application by TCP. It is returned to the caller of the CONNECT service. This field is an unsigned positive integer with a maximum value of 65,534. Default: None
ACMRPORT	Remote well-known service port. Indicates the remote transport layer address or port. This value represents the TCP port on the remote host to which the client application is trying to connect. It must be filled in by the calling client application unless the ACMSRVCE field is specified. If ACMSRVCE is specified, ACMRPORT will be filled in with the resolved remote port number before returning to the caller of the CONNECT service. This field is an unsigned positive integer with a maximum value of 65,534. Default: None

024040-200100

PARAMETER	DESCRIPTION
ACMSRVCE	Transport layer service name. Indicates the local transport layer service name. This string is mapped into a transport layer address or port, through internal Domain Name Resolution (DNR). The resolved value will be the well-known port to which the CONNECT service attempts to establish a connection. This field has a maximum length of 36 bytes. This field is optional and is not modified by the CONNECT service. Default: None
ACMOPTNS	TCP connection initialization options. ◆ ACMNODNR – DNR Suppression option. Skip internal DNR calls to resolve and return the remote IP address into an IP name in the ACMRNAME field. If an application is designed such that TCP connection establishment and release happens frequently, this option can save processing time. ◆ ACMLTRAN – Client-Data Listener option. This option is for the LISTEN service and is not validated or modified by the CONNECT service. ◆ ACMSYNC – Listen Syncport option. This option is for the LISTEN service and is not validated or modified by the CONNECT service. Default: None
ACMLADDR	Local IP host address. Indicates the local host internet address. This field is an unsigned four-byte integer value. The local host internet address is updated when a server connection is established, and is returned to the caller. Default: None
ACMRADDR	Remote IP host address. Indicates the remote host internet address. Either this field or the remote host name (ACMRNAME) field must be specified. This field is an unsigned four-byte integer value. The remote host internet address is updated when a server connection is established, and is returned to the caller. Default: None
ACMLNAME	Local IP host name. Indicates the local host internet name. This field is a 255-byte character string that is padded with blanks. The local host internet name is updated when a client connection is established, and is returned to the caller. Default: None
ACMRNAME	Remote IP host name. Indicates the remote host internet name. Either this value or the remote IP address (ACMRADDR) field must be specified. This is a 255-byte character string that is padded with blanks. The remote host internet name is updated when a server connection is established, and is returned to the caller. Default: None
ACMTIMEO	Client-Data Listener timeout value. This field is optionally used by the LISTEN service and is not validated or modified by the CONNECT service. Default: 1 second

200801-024040-200100

Network Considerations

The ACM is a common data structure used for both client and server connection initialization. There are common and unique values specified for a particular service request.

This table describes network considerations for Assembler API:

NAME	SERVER CONDITIONS FOR LISTEN	CLIENT CONDITIONS FOR CONNECT
ACMTLSTN	Listen token returned to user application.	
ACMTRNID	Listen START transaction ID.	
ACMLPORT	Local server or listening transport provider well-known port selected by user application.	Local assigned transport provider port returned to user application.
ACMRPORT	Remote client transport provider port returned to user application.	Remote server transport provider well-known port selected by user application.
ACMSRVCE	Local server transport provider service name selected by user application.	Remote server transport provider service name selected by user application.
ACMRADDR	Remote IP host address returned to user application.	Remote IP host address selected or returned to user application.
ACMLNAME	Local IP host name returned to user application.	Remote IP host name returned to user application.
ACMRNAME	Remote IP host name returned to user application.	Local IP host name returned to user application.
ACMTIMEO	Client-Data Listener timeout value.	

Completion Information

The CONNECT service completes normally when a connection with a server is established. The CONNECT service initializes the client environment with the transport provider (API) and actively contacts a server and update connection information within the ACM. Establishing a client connection is represented by storage and is referred to as the token. When a connection is successfully established the ACM is updated with information related to the connection.

The ACM is updated with information related to the new connection. The local and remote port, IP address, and host names are resolved. Negotiated transport provider SEND and RECEIVE buffering values are returned. Either the ACM return code (ACMRTNCD) or register 15 should be checked to determine the success or failure of the CONNECT service. A zero (0) return code indicates a successful connection.

The return and diagnostic codes should be interpreted by the application to determine the reason for failure. Errors indicating CPT, the transport provider (API), or CICS termination are minor. Errors should be interrogated for level of severity.

20 024040-200100

Return Codes

The CONNECT service returns a code in registers 15 and 0 that indicates the results of the execution. These values are in the ACMRTNCD (R15) and ACMDGNCD (R0) within the ACM. The diagnostic code generally indicates the transport provider return code. Read **Appendix C – MESSAGES AND CODES** in the ***CICS Programmer's Toolkit Installation and Administration Guide*** for the return code cross reference table.

This table describes the CONNECT service return codes:

RETURN	DIAGNOSTIC CODE	DESCRIPTION
CPTIRCOK		Successful.
CPTEVERS		Control block version number is not supported.
CPTECONN	Yes	Requested host/service/port is not found.
CPTEPROT		Specified protocol is not supported.
CPTENAPI	Yes	Transport provider API is not available.
CPTETERM	Yes	Environment is being terminated.
CPTERLSE	Yes	Release indication.
CPTEDISC	Yes	Disconnect indication.
CPTEPRGE	Yes	CPT Interface terminating.
CPTEINTG	Yes	Transport provider API integrity error.
CPTEENVR	Yes	Transport provider API environment error.
CPTEFRMT	Yes	Transport provider API format error.
CPTEPROC	Yes	Transport provider API procedure error.
CPTABEND		Abnormal exception occurred.
CPTEOTHR	Yes	An undefined exception occurred.

200801-024040-200100

Usage Information

The CONNECT service lets user-written application programs implement TCP/IP client facilities. The CONNECT service generalized parameter list (ACM) describes the application's communications requirements as well as information related to established connections. On completion, the ACM contains fields initialized by both a user application and by the CONNECT service.

There are required and optional fields initialized by a user or calling application. The ACM version number is required. The server must be identified by the calling program. The server is specified by selecting the remote IP address (ACMRADDR) or host name (ACMRNAME) fields, and the remote port (ACMRPORT) or service name (ACMSRVCE). The selection of port or service name defines the server well-known port address. Optional fields control data transfer buffering, statistics, tracing, and subtask initialization.

On completion of the CONNECT service, the ACM contains information related to the established connection. A token which identifies the connection is returned in the ACM, and must be used in all subsequent requests that refer to the connection. The user application program should make no assumptions regarding the format of a token, other than that it is an unsigned, full word value. Information related to the negotiated buffer values, host names, host addresses, and transport provider addresses are returned in the ACM.

The version number (ACMVERS) indicates the CPT release level in which this user application program is written. This required field must be set to a binary 2 and is validated by the CONNECT service before processing the request.

The function code (ACMFUNC) indicates the CPT callable service ID. The field is initialized by the CPT service stub program and has little value to the application except for dump analysis. The function code can identify and map an argument with the error or trace logs, and dump analysis.

The remote IP address (ACMRADDR) or remote host name (ACMRNAME) is required. These fields identify the host to which the CONNECT service initiates a connection request. The IP address has precedence over host name. This implies that the host name field is only used if a IP address is not specified.

The transport provider port number (ACMRPORT) or service name (ACMSRVCE) is required. These fields identify the well-known port to which the CONNECT service will initiate a connection request. The port number has precedence over service name. This implies that the service name field is only used if a port number is not specified.

User application programs have the ability to control CPT and transport provider data transfer buffering. The ACMQSEND, ACMMSEND, ACMQRECV, and ACMMRECV specify the number and size of buffers allocated. The SEND and RECEIVE buffers are allocated on initial entry into either the SEND or RECEIVE service. The corresponding values used by the SEND and RECEIVE services are independent of each other. The SEND service multiplies the queue and buffer values to determine output storage requirements. The RECEIVE service performs a similar function to determine input storage requirements. The product of the queue and buffer values cannot exceed 61,440. CPT requires some additional storage to manage these buffers. This extra storage is included in the allocation.

20-024040-200100

The SEND service uses the ACMQSEND value; the RECEIVE service uses the ACMQRECV value. These values indicate the maximum number of uncompleted SEND or RECEIVE requests that can be queued by the application to the transport provider. The CPT data transfer services schedule asynchronous transport provider or API requests on the caller's behalf. These API requests must be completed before they can be used again. If the queue values are too small, the CPT data transfer service may block the caller's request and schedule a WAIT command within the service routines. If the queue values are too large, the user application may be wasting storage.

The SEND service uses the ACMMSEND value; the RECEIVE service uses the ACMMRECV value. These values indicate the maximum number of user data bytes that can be transferred by the application in a single SEND or RECEIVE request to the transport provider. The user application is not limited to these values within the data transfer services. However, it is important to note that multiple transport provider or API requests are issued to complete the caller's request. Information on queue and buffer size can be found in the SEND and RECEIVE service description section in this chapter.

Initially, the tuning of data transfer storage may not be a concern; however, the ability to control storage allocation can prove beneficial to the application or CICS region. Additionally, queue size can increase data transfer throughput. Consider enabling the statistics option to gather CPT statistical information, which can be used to set the SEND or RECEIVE queue and buffer size values.

The CONNECT service can modify data transfer buffer allocation values. These values are negotiated with the transport provider and, depending on the site configuration, can be reduced. Any application dependent on these values should check them on return. These values will generally not be modified when giving reasonable numbers. However, it is advisable to check with the site administrator for maximum values for the API transport services.

A number of arguments are not set by the calling application, but are returned to the caller. These values represent information related to the client connection and can be used by the application. The local port, host name, and IP address are returned as well as the client's corresponding values.

200801-024040-200100

Example:

In this example a simple client ACM is built and the CONNECT service request is performed. The application program sets the argument version number to 1 and the remote to 1234. Control is returned from the CONNECT service on establishment of a connection, or else by some error occurring. Register 15 is tested to determine the success of the request. If register 15 is non-zero, an error has occurred and the diagnostic code will indicate the reason for failure. If register 15 is zero, then the CONNECT service completed successfully and a token representing the data transfer connection is returned. The token is used for all CPT requests related to that connection.

```
*
*          Dsect's
*
           T09DACM MF=DSECT     Argument for Connection Management
*
*          Working storage
*
DFHEISTG   DSECT
           .
           .
           .
CONNARG    DS     XL(ACMLEN)    Argument for Connection Management
*
*          Entry
*
label      DFHEIENT
           .
           .
           .
*
*          CPT Connection Management request
*          LA     R2,CONNARG    Load Argument for Connection Management
           USING  T09DACM,R2
           MVC    ACMVERS,=H'2'    Set Version number
           MVC    ACMRPORT,=H'1234'Set Server well-known port
           MVC    ACMRNAME(9),=C'LOCALHOST' Set Server Host name
           LR     R1,R2            Load ACM address in Reg 1
           L      R15,=V(T09FCONN) Load CONNECT service Stub addr.
           BALR   R14,R15          Issue CONNECT service
           LTR    R15,R15          Test Return Code
           BZ     LOADTOKN         Good, process data
           L      R4,ACMRTNCD      Load Return Code
           L      R5,ACMDGNCD      Load Diagnostic Code
           .
           . Process and log Connection Management request error
           .
           B      END              Termination Transaction
LOADTOKN   DS     0H
           L      R6,ACMTOKEN      Load Connection Token
           DROP   R2                  ACM
           .
           . Application and CPT Data Transfer (SEND/RECEIVE) processing
           .
           .
           . CPT Connection Release
           .
*
*          Terminate Transaction
*
END        DS 0H
           EXEC   CICS RETURN
```

Example:

In this example a simple client ACM is built and the CONNECT request is performed. The application program will connect to whatever port is mapped into service name ECHO. The T09MCALL assembler macro instruction sets the argument version number and call the CONNECT service. Control is returned from the CONNECT service on establishment of a connection or by some error. Register 15 is tested to determine the success of the request. If register 15 is non-zero, an error has occurred and the diagnostic code will indicate the reason for failure. If register 15 is zero, then the CONNECT service completed successfully and a token representing the data transfer connection is returned. The token is used for all CPT requests related to that connection.

```
*
*         Dsect's
*
          T09DACM MF=DSECT    Argument for Connection Management
*
*         Working storage
*
DFHEISTG  DSECT
          .
          .
          .
CONNARG   DS     XL(ACMLEN)   Argument for Connection Management
*
*         Entry
*
label  DFHEIENT
          .
          .
          .
*
*         CPT Connection Management request
*
          LA     R3,CONNARG   Load Argument for Connection Management
          USING  T09DACM,R3
          MVC    ACMSRVCE(4),=C'ECHO'    Set Server Service name
          MVC    ACMRNAME(9),=C'LOCALHOST'  Set Server Host name
          T09MCALL CONNECT,PARM=CONNARG Issue CONNECT Service
          LTR    R15,R15                Test Return Code
          BZ     LOADTOKN               Good, process data
          L      R4,ACMRTNCD            Load Return Code
          L      R5,ACMDGNCD            Load Diagnostic Code
          .
          . Process and log Connection Management request error
          .
          B      END                    Termination Transaction
LOADTOKN  DS     0H
          L      R6,ACMTOKEN            Load Connection Token
          DROP   R3                        ACM
          .
          . Application and CPT Data Transfer (SEND/RECEIVE) processing
          .
          .
          . CPT Connection Release
          .
*
*         Terminate Transaction
*
END       DS     0H
          EXEC   CICS RETURN
```

200801-024040-200100

GIVE

The GIVE service releases ownership of a connection and associated internal CPT resources. The GIVE service is optional and does not affect an active connection, nor does it issue any transport provider requests. This service affects CPT TRUE management routines, scheduled on the user's behalf during task termination.

To invoke the GIVE service, a user application is required to first build an Argument for Facility Management (AFM) and then to issue a call to the GIVE routine. The only valid and required arguments are the AFM version number and the connection token. On completion, a return code is set to indicate the success or failure of the request.

This table describes the GIVE service arguments:

MACRO ID	DSECT NAME	SIZE	CREATED BY
T09DAFM	AFM	34 (X'22')	User application or the LISTEN service.

Assembler Data Area

This is what the DSECT control block looks like in Assembler language:

```
Name      Operation Operands  Description

AFM       DSECT     ,
AFMVERS   DS        H         Version number
AFMFUNC   DS        H         Function code
AFMTOKEN  DS        A         Token (CEP)
          DS        A         Reserved
          DS        F         Reserved
AFMRTNCD  DS        F         Return code
AFMDGNCD  DS        F         Diagnostic code
AFMOPTNS  DS        0F        Facility management option codes
AFMOPCD4  DS        X         Option 4
AFMOPCD3  DS        X         Option 3
AFMOPCD2  DS        X         Option 2
AFMOPCD1  DS        X         Option 1
          DS        H         Reserved
```

PARAMETER	DESCRIPTION
AFMVERS	Version Indicates the CPT version number of the argument list used by the calling program. This required field must be set to a binary 2 for this release of CPT. Default: None

PARAMETER	DESCRIPTION
AFMFUNC	Function code Indicates the function or callable service ID requested by the application program. This field is set by the application, but is initialized by the TRUE interface stub program. Default: None
AFMTOKEN	Connection or endpoint token Specifies a token that represents a TCP connection, a TCP listening endpoint, or a UDP endpoint. A token is created by the TCP connection initiation routines or by the UDP data transfer and endpoint creation routines. The token is required. Default: 0
AFMRTNCD	Return code Indicates the return code set by the GIVE service. This value is also returned in register 15 and indicates the success or failure of the service. Default: 0
AFMDGNCD	Diagnostic code Indicates the diagnostic code received by the GIVE service for a transport provider request and is not set by the GIVE service. The GIVE service does not issue transport provider requests; hence, it never sets the diagnostic code. Default: 0
AFMOPTNS	Options and events Specifies GIVE processing control options or events detected by the GIVE service. Currently, this facility is reserved for internal processing. Default: 0

Completion Information

The GIVE service completes normally when all resources associated with this connection are processed.

On normal return to the application program, the general return code in register 15 (AFMRTNCD) is set to zero (CPTIRCOK). The diagnostic code in register 0 (AFMDGNCD) is always zero.

If the GIVE service completes abnormally, some resources associated with this connection cannot be successfully transferred from one task to another. The general return code (AFMRTNCD) in register 15 and the diagnostic code (AFMDGNCD) in register 0 indicate the nature of the failure. The diagnostic code (AFMDGNCD) is not used by the GIVE service and no information is returned.

Return Codes

The GIVE service returns a code in register 5 that indicates the result of the execution. This value can be found in the AFMRTNCD (R15) within the AFM. Read **Appendix C – MESSAGES AND CODES** in the ***CICS Programmer's Toolkit Installation and Administration Guide*** for the return rode cross reference table.

This table describes the GIVE service return codes:

RETURN	DESCRIPTION
CPTIRCOK	Successful.
CPTEVERS	Control block version number is not supported.
CPTETOKN	Specified token is not valid.
CPTENAPI	Transport provider API is not available.
CPTABEND	Abnormal exception occurred.
CPTEOTHR	An undefined exception occurred.

Usage Information

The GIVE service releases ownership of a connection. This service is non-blocking and does not affect any pending transport provider data transfer requests. Disassociating resources from a task lets the CPT properly manage resources during task termination. This ability to GIVE and TAKE ownership of connections offers the user a range of programming options, while still providing CPT with resource management capabilities.

The GIVE service requires the application to set the AFM version number and token fields. No other fields are referenced.

Note: When a connection is established there are internal CPT resources associated with that connection. CPT is responsible for proper clean-up of those resources on task or transaction termination. These resources include storage allocated by CPT, the API, and the transport provider storage.

A server application is a good example of how the GIVE service benefits a user application. A listening task issues the GIVE service and starts a new transaction to handle data transfer. The data transfer transaction then TAKEs the connection. This sequence would prevent a connection from being closed (implicitly by the CPT task termination exit) if the server application terminates. However, if the data transfer transaction is terminated without issuing an explicit close (CPT CLOSE service) an implicit close is scheduled, and resource management is handled by the CPT task termination exit.

024040-200100

The version number (AFMVERS) indicates the CPT release level in which this user application program is written. This required field must be set to a binary 2 and is validated by the GIVE service before processing the request.

The function code (AFMFUNC) indicates the CPT callable service ID. The field is not initialized by a user application program and has little value to the application except for dump analysis. The function code can identify and maps an argument list with the error or trace log and dump analysis.

The token (AFMTOKEN) indicates the connection and internal resources to be processed by the GIVE service. This is a required field and is validated by the GIVE service before processing request.

The AFMOPTCD field specifies GIVE service processing control options, and provides a mechanism for event notification on return to the application program. Currently, this field is not used by application programs.

200801-024040-200100

Example:

This example establishes a server data transfer connection, issues the GIVE service, and starts a data processing transaction. The token is loaded from the ACM and is used by the GIVE service. The AFM is initialized with the version number and token before the GIVE service is called. On completion of the GIVE service, register 15 is checked and, if successful, processing continues.

```
*
*         Dsect's
*
          T09DACM MF=DSECT    Argument for Connection Management
          T09DAFM MF=DSECT    Argument for Facility Management
*
*         Working storage
*
DFHEISTG  DSECT
          .
          .
LSTNARG   DS      XL(ACMLEN)    Argument for Connection Management
GIVEARG   DS      XL(AFMLEN)    Argument for Facility Management
*
*         Entry
*
label     DFHEIENT

      .
      .

LISTEN    DS      0H
          .
          . CPT Listen (Server) Connection Management request
          .
          L       R6,ACMTOKENLoad CPT Token

          .
          .

*
*         CPT Facility Management GIVE service request
*
          LA      R4,GIVEARG    Load Argument for Facility Management addr
          USING T09DAFM,R4
          MVC     AFMVERS,=H'2' Set Version number
          ST      R6,AFMTOKEN   Save FM connection Token
          LR      R1,R4         Load AFM address in Reg 1
          L       R15,=V(T09FGIVE)Load GIVE service Stub address
          BALR    R14,R15    Issue GIVE service
          LTR     R15,R15    Test Return Code
          BZ      START      Good, process connection
*
*         GIVE Facility Management Error
*
          L       R4,AFMRTNCD   Load Return Code
          L       R5,AFMDGNCD   Load Diagnostic Code
          .
          . Log and process error
          .
          B       CLOSE         Terminate Server and Transaction
```

SC24-5404-02 20040-200100

```
*
(          Issue CICS Start for Data Processing Task
*
START      DS     0H     .
           EXEC CICS START TRANSID(trans-id) FROM(AFMTOKEN)
           .
           B      LISTEN        Listen for more Client connection
CLOSE         DS  0H
           .
           . CPT Connection Release
           .
*
*          Terminate Transaction
*
END        DS     0H
           EXEC   CICS RETURN
```

Example:

This example establishes a server data transfer connection, issues the GIVE service, and starts a data processing transaction. The token is loaded from the ACM and is used by the GIVE service. The AFM is initialized with the token. This example differs from example 1 in that the T09MCALL sets the version number and issues the GIVE call. On completion of the GIVE service, register 15 is checked and, if successful, processing continues.

```
*
*         Dsect's
*
          T09DACM MF=DSECT     Argument for Connection Management
          T09DAFM MF=DSECT     Argument for Facility Management
*
*         Working storage
*
DFHEISTG  DSECT
          .
          .
          .
LSTNARG   DS  XL(ACMLEN)       Argument for Connection Management
GIVEARG   DS  XL(ACMLEN)       Argument for Facility Management
*
*         Entry
*
label     DFHEIENT
          .
          .
          .
LISTEN    DS     0H
          .
          . CPT Listen (Server) Connection Management request
          .
          L      R9,ACMTOKEN     Load Data Transfer Token
          .
          .
          .
*
*         CPT Facility Management GIVE service request
*
          LA     R7,GIVEARGLoad Argument for Facility Management addr
          USING T09DAFM,R7
          ST     R9,AFMTOKEN     Save FM connection Token
          T09MCALL GIVE,PARM=GIVEARGIssue GIVE service
          LTR    R15,R15         Test Return Code
          BZ     START           Good, process connection
*
*         GIVE Facility Management Error
*
          L      R4,AFMRTNCD     Load Return Code
          L      R5,AFMDGNCD     Load Diagnostic Code
          .
          . Log and process error
          .
          B      CLOSE       Terminate Server and Transaction
```

```
*
*          Issue CICS Start for Data Processing Task
*
START      DS     0H     .
           EXEC CICS START TRANSID(trans-id) FROM(AFMTOKEN)
           .
           B      LISTEN       Listen for more Client connection
CLOSE      DS     0HC


           .
           . CPT Connection Release
           .
*
*          Terminate Transaction
*
END        DS     0H
           EXEC   C            ICS RETURN
```

200801-024040-200100

LISTEN

This service provides a server facility for use by an application program. The LISTEN service establishes a session with the local transport provider, passively listens for connection requests, then accepts new connections. When connection with a client is established, the LISTEN service either returns control to the calling program or starts a defined transaction. Information related to the connection is updated and returned within the ACM.

To invoke the LISTEN service, a user application is required to first build an ACM, then issue a call to the LISTEN routine. The minimum information required by this service is the version number and either the local transport provider port or the service name. Optional information related to data transfer buffering, CPT statistics and tracing, and subtask initialization can be specified. Completion of a LISTEN service depends on options selected within the ACM.

This table describes the LISTEN service arguments:

MACRO ID	DSECT NAME	SIZE	CREATED BY
T09DACM	ACM	676 (X'2A4')	User application or common area obtained by a RETRIEVE command from a started transaction.

Assembler Data Area

This is what the DSECT control block looks like in Assembler language:

```
Name      Operation     Operands   Description

ACM       DSECT         ,
ACMVERS   DS            H          Version number
ACMFUNC   DS            H          Function code
ACMTOKEN  DS            A          Token (CEP)
          DS            A          Reserved
          DS            F          Reserved
ACMRTNCD  DS            F          Return code
ACMDGNCD  DS            F          Diagnostic code
ACMSTATS  DS            0F         Statistics flags
          DS            XL3
ACMSTAT   DS          X          Primary Statistics request byte
ACMSCONN  EQU         X'01'       - Connection statistics
ACMSTERM  EQU         X'02'       - Termination statistics

ACMTRACE  DS          0F         Trace flag
          DS          XL2
ACMTRAC2  DS          X          Second Trace byte
*                                   (for high level lang)
ACMTTKNS  EQU         X'01'       - Trace token information
ACMTTPL   EQU         X'02'       - Trace TPL block
ACMTRLSE  EQU         X'04'       - Trace release information
ACMTSTOR  EQU         X'08'       - Trace GETMAIN/FREEMAIN
ACMTCLTD  EQU         X'10'      - Trace TD from client (IBM style)
ACMTRAC1  DS          X          First Trace byte
*                                   (for high level lang)
ACMTNTRY  EQU         X'01'       - Trace entry points
```

```
ACMTARGS  EQU       X'02'      - Trace arguments
ACMTRECV  EQU       X'04'      - Trace TRECV
ACMTSEND  EQU       X'08'      - Trace TSEND
ACMTTERM  EQU       X'10'      - Trace termination
ACMTPASS  EQU       X'20'      - Trace TAKE
ACMTCLSE  EQU       X'40'      - Trace CLOSE
ACMTTERR  EQU       X'80'      - Trace TPL errors

ACMQSEND  DS        F          TSEND queue size
ACMMSEND  DS        F          Maximum TSEND TPL buffer size
ACMQRECV  DS        F          TRECV queue size
ACMMRECV  DS        F          Maximum TRECV TPL buffer size
ACMTLSTN  DS        F          LISTEN Token
ACMUCNTX  DS        F          User context field
ACMTRNID  DS        CL4        Transaction ID
          DS        X          Reserved for C string
          DS        XL3        Unused - available
ACMLPORT  DS        H          Local transport provider port
ACMRPORT  DS        H          Remote transport provider port
ACMSRVCE  DS        CL36       Local/remote service name
          DS        X          Reserved for C string
          DS        X          Unused - available
ACMOPTNS  DS        OH         ACM option flags
ACMOPTN2  DS        X          ACM option flag 2
ACMOPTN1  DS        X          ACM option flag 1
ACMNODNR  EQU       X'04'       - No local/remote name resolution
ACMLTRAN  EQU       X'02'       - LISTEN start transaction
*                                 specified in first 1-4 bytes of
*                                 client data
ACMSYNC   EQU       X'01'       - Issue syncpoint for LISTEN
ACMLADDR  DS        F          Local IP host address
ACMRADDR  DS        F          Remote IP host address
ACMLNAME  DS        CL255      Local IP host name
          DS        X1         Reserved for C string
ACMRNAME  DS        CL255      Remote IP host name
          DS        X          Reserved for C string
          DS        H          Reserved
          DS        H          Reserved
ACMTIMEO  DS        F          Timeout to receive client data
          DS        F
```

PARAMETER	DESCRIPTION
ACMVERS	Version Indicates the version number of the CPT argument used by the calling program. This required field must be set to a binary 2 for this release of CPT. Default: None
ACMFUNC	Function code Indicates the function or callable service requested by the application program. This field is not set by the application, but is initialized by the TRUE interface stub program. Default: None (generated by service stub)
ACMTOKEN	TCP connection token Specifies the token created and returned by the LISTEN service. It will be used in all subsequent calls for the client application. Default: 0 (token returned)

PARAMETER	DESCRIPTION
ACMRTNCD	Return code Indicates the return code set by the LISTEN service. This value is also returned in register 15 and indicates the success or failure of the service. Default: 0
ACMDGNCD	Diagnostic code Indicates the diagnostic code received by the LISTEN service for a transport provider request. There is a detailed explanation of this value in the transport provider's API Programmer's Reference Guide. Default: 0
ACMSTATS	This field specifies statistics logging options for the application program. The facility can be used for debugging and tuning during development. ◆ ACMSCONN – Specifies that a message(s) be generated on establishment of either a listen service or a data transfer connection. These messages are generated by the CPT LISTEN and CONNECT services. The message numbers associated with this option are CPT802I and CPT803I. ◆ ACMSTERM – Specifies that a message(s) be generated on termination of an established connection. These messages are generated by the CPT CLOSE service. The message numbers associated with this option are CPT807I, CPT808I, and CPT809I. Default: 0 (no statistics logging)

024040-200100

PARAMETER	DESCRIPTION
ACMTRACE	This field specifies trace logging options for the application program. The facility can be used for debugging during development. ◆ ACMTNTRY – Specifies that a message be generated on entry to a CPT service routine. The message is generated by all CPT service routines. The message number associated with this option is CPT901I. ◆ ACMTARGS – Specifies that a hex dump of the caller's arguments be generated on entry and exit of a CPT service routine. Messages are generated by the corresponding service routine. The message numbers associated with this option are CPT902I, CPT903I, CPT904I, CPT911I, CPT912, CPT921I, CPT922I, CPT924I, CPT925I, CPT930I, and CPT932I. ◆ ACMTRECV – Specifies that a hex dump be logged of the transport provider's input (RECEIVE) data. Messages are generated by the CPT RECV service. The message number associated with this option is CPT913I. ◆ ACMTSEND – Specifies that a hex dump be logged of the transport provider's output (SEND) data. Messages are generated by the CPT SEND service. The message number associated with this option is CPT914I. ◆ ACMTTERM – Specifies that a message be generated on termination of a CPT service routine. Messages are generated by all CPT service routines. The message number associated with this option is CPT908I. ◆ ACMTPASS – Specifies that a hex dump of resources related to a passed connection be logged. Messages are generated by the CPT GIVE and TAKE services. The message numbers associated with this option are CPT919I and CPT920I. ◆ ACMTCLSE – Specifies that a hex dump of resources related to a CLOSE processing of a connection be logged. Messages are generated by the CPT CLOSE service. The message number associated with this option is CPT906I ◆ ACMTTERR – Specifies that a hex dump be logged of a transport provider API parameter list that fails successful completion. The message number associated with this option is CPT400I. ◆ ACMTTKNS – Specifies that a hex dump of the connection token be logged. Messages are generated by all services routines on entry. The message number associated with this option is CPT909I. ◆ ACMTTTPL – Specifies that a hex dumped logged of the transport provider API parameter list. The message numbers associated with this option are CPT917I and CPT935I. ◆ ACMTRLSE – Specifies that a message indicating a transport provider release indication be logged. Messages are generated by various CPT service routines. The message number associated with this option is CPT414I. ◆ ACMTSTOR – Specifies that a hex dump of storage management argument be logged. Messages are generated by various CPT service routines. The message numbers associated with this option are CPT928I and CPT929I. ◆ ACMTCLTD – Trace transient data writes from the LISTEN service (used with the ACMLTRAN listen start transaction option. The message number associated with this option is CPT918I. Default: 0 (no trace logging)

200801-024040-200100

PARAMETER	DESCRIPTION
ACMQSEND	API send queue size Specifies the maximum number of uncompleted SEND requests that can be queued by the application to the transport provider (API). This value lets applications control output processing and can affect throughput rates. The value is negotiated with the transport provider and can be modified by the transport provider. The total allocation for output processing is the product of the SEND queue and buffer size values and cannot exceed 61K. Default: 4
ACMMSEND	API send buffer size Specifies the maximum number of user data bytes that can be transferred by the application in a single SEND request to the transport provider (API). This value lets applications control output processing and can affect throughput rates. The value is negotiated with the transport provider and can be modified by the transport provider. The total allocation for output processing is the product of the SEND queue and buffer size values and cannot exceed 61K. Default: 4096
ACMQRECV	API receive queue size Specifies maximum number of uncompleted RECEIVE requests that can be queued by the application to the transport provider (API). This value lets applications control input processing and can affect throughput rates. The value is negotiated with the transport provider and can be modified by the transport provider. The total allocation for input processing is the product of the RECEIVE queue and buffer size values and cannot exceed 61K. Default: 4
ACMMRECV	API receive buffer size Specifies the maximum number of user data bytes that can be transferred by the application in a single RECEIVE request to the transport provider (API). This value lets applications control input processing and can affect throughput rates. The value is negotiated with the transport provider and can be modified by the transport provider. The total allocation for input processing is the product of the RECEIVE queue and buffer size values and cannot exceed 61K. Default: 4096
ACMTLSTN	Listen service token statistics Specifies the token used by the LISTEN service. This token is not available for data transfer. The only valid function that can be performed is a CLOSE request for long running active listeners. Generally, this value is not used by the application unless an explicit call to the CLOSE service is required. Read the description for ACMTOKEN (earlier in this table) for all other services. Default: 0 (token returned)
ACMUCNTX	One word of user context Specifies one arbitrary word of user context to be associated with the connection. The information provided is not interpreted by CPT, and is merely saved with other connection information. Default: 0 (no user context)
ACMTRNID	Listen start transaction ID A four-byte character string that the LISTEN service starts on successful establishment of a new connection. If TRANSID is specified, the LISTEN server loops for new connections and does not return to the calling program until CICS, CPT, or transport provider (API) termination. This field is optional and is not modified by the listen service. This field should not be specified if the ACMLTRAN option and ACMTIMEO value are specified. Default: None

PARAMETER	DESCRIPTION
ACMLPORT	Listen well-known service port Indicates the local transport layer address or port. This value represents the well-known port on which a server application will listen for connection requests. Either this value or the transport layer service name (ACMSRVCE) must be specified. This field is an unsigned positive integer with a maximum value of 65,534. The value must be unique for each server application. Default: None
ACMRPORT	Remote well-known service port Indicates the remote transport layer address or port. This value represents the client port number. This value is returned to the caller. This field is an unsigned positive integer with a maximum value of 65,534. Default: None
ACMSRVCE	Transport layer service name Indicates the local transport layer service name. This string is mapped into a transport layer address or port, through internal DNR. The resolved value is the well-known port to which the client application connects. This field has a maximum length of 36 bytes. The value must be unique for each server application. This field is optional and is not modified by the LISTEN service. Default: None
ACMOPTNS	TCP connection initialization options ◆ ACMNODNR – DNR Suppression option. Skips internal DNR calls to resolve and return the remote IP address into an IP name in the ACMRNAME field. If an application is designed such that TCP connection establishment and release happens frequently, this option can save processing time. ◆ ACMLTRAN – Client-Data Listener option. Specifies that the Listen call will receive the input datastream to determine the transaction ID to be started. See **Client-Data Listener** option for the required input formats. This option must be used with ACMTIMEO, and should not be used with ACMTRANID. ◆ ACMSYNC – Listen Syncpoint option. Issues a CICS syncpoint before starting any transaction from the LISTEN service. Default: None
ACMLADDR	Local IP host address Indicates the local host internet address. This field is an unsigned four-byte integer value. The local host internet address is updated on establishment of a client connection, and is returned to the caller. Default: None
ACMRADDR	Remote IP host address Indicates the remote host internet address. This field is an unsigned four-byte integer value. The remote host internet address is updated on establishment of a client connection, and is returned to the caller. Default: None
ACMLNAME	Local IP host name Indicates the local host internet name. This field is a 255-byte character string that is padded with blanks. The local host internet name is updated on establishment of a client connection, and is returned to the caller. Default: None

200801-024040-200100

PARAMETER	DESCRIPTION
ACMRNAME	Remote IP host name Indicates the remote host internet name. This field is a 255-byte character string that is padded with blanks. The remote host internet name is updated on establishment of a client connection, and is returned to the caller. Default: None
ACMTIMEO	Client-Data Listener timeout values Specifies the maximum number of seconds that a Listener can wait to receive the client datastream when the ACMLTRAN option is specified. Default: 1 second

Network Considerations

The ACM is a common data structure used for both client and server connection initialization. There are common and unique values specified for a particular service request.

This table describes network considerations for Assembler API:

NAME	SERVER CONDITIONS FOR LISTEN	CLIENT CONDITIONS FOR CONNECT
ACMTLSTN	Listen token returned to user application.	
ACMTRNID	Listen START transaction id.	
ACMLPORT	Local server or listening transport provider well-known port selected by user application.	Local assigned transport provider port returned to user application.
ACMRPORT	Remote client transport provider port returned to user application.	Remote server transport provider well-known port selected by user application.
ACMSRVC	Local server transport provider service name selected by user application.	Remote server transport provider service name selected by user application.
ACMRADDR	Remote IP host address returned to user application.	Remote IP host address selected or returned to user application.
ACMLNAME	Local IP host name returned to user application.	Local IP host name returned to user application.
ACMRNAME	Remote IP host name returned to user application.	Remote IP host name selected or returned to user application.
ACMTIMEO	Client-Data Listener timeout value.	

Completion Information

Completion of a request to the LISTEN service depends on the arguments selected. If no transaction ID is specified, the LISTEN service returns control to the calling program when connection with a client is established. The caller's argument list is updated with information related to the new connection. If a transaction ID is specified, the LISTEN service does not return control to the calling program until a failure is detected. The caller's argument list is generally not updated, with exception to the return code information.

The LISTEN service initializes the server environment with the transport provider (API), waits for a connection request, establishes a connection with the client, and updates connection information within the ACM. Establishing a listening connection and a client connection are represented by storage and are referred to hereafter as tokens. Establishing a client connection updates the ACM with information relative to the connection. The information is returned to the user or is passed to the data processing transaction.

The server application contains two tokens representing endpoints to the transport provider. The first token (ACMTOKEN) represents the client connection and is used for data transfer. The other token (ACMTLSTN) represents the listening connection. This listening connection can only be referenced within the CPT CLOSE service. This lets an explicit ability close a server or listening connection. All other CPT services performed with the LISTEN token fail with an invalid token. Implicit clean-up of the LISTEN token is provided by the TRUE interface. Therefore, an explicit call to the CLOSE service is not required.

When the LISTEN service is initiated without a transaction ID, control is returned to the calling program when a connection with a client is established. The caller's argument list is updated with information related to the new connection. The local and remote port, IP address, and host names are resolved. Negotiated transport provider SEND and RECEIVE buffering values are returned. Either the ACM return code (ACMRTNCD) or register 15 must be checked to determine the success or failure of LISTEN service. A zero (0) return code indicates a successful client connection is established.

When the LISTEN service is initiated with a transaction ID, it operates as a CICS long running task. The LISTEN service establishes client connections and starts a data processing transaction. The data processing transaction receives a copy of the connection management argument. The data transfer or client connection token is derived from the ACMTOKEN field. After the new transaction is initiated the LISTEN service continues waiting for new client connections. The LISTEN service continues to listen and start client connections until an error occurs.

The return and diagnostic codes should be interrogated to determine the reason for failure. Errors indicating CPT, the transport provider (API), or CICS termination are minor. Errors should be interrogated for level of severity.

200R01-024040-200100

Return Codes

The LISTEN service returns a code in registers 15 and 0 that indicates the results of the execution. These values are in the ACMRTNCD (R15) and ACMDGNCD (R0) within the ACM. The diagnostic code is optional and indicates the transport provider return code. Read **Appendix C – MESSAGES AND CODES** in the ***CICS Programmer's Toolkit Installation and Administration Guide*** for the return code cross reference table.

This table describes the LISTEN service return codes:

RETURN	DIAGNOSTIC CODE	DESCRIPTION
CPTIRCOK		Successful.
CPTEVERS		Control block version number is not supported.
CPTECONN	Yes	Requested host/service/port is not found.
CPTEPROT		Specified protocol is not supported.
CPTENAPI	Yes	Transport provider API is not available.
CPTETERM	Yes	Environment is being terminated.
CPTERLSE	Yes	Release indication.
CPTEDISC	Yes	Disconnect indication.
CPTEPRGE	Yes	CPT Interface terminating.
CPTEINTG	Yes	Transport provider API integrity error.
CPTEENVR	Yes	Transport provider API environment error.
CPTEFRMT	Yes	Transport provider API format error.
CPTEPROC	Yes	Transport provider API procedure error.
CPTABEND		Abnormal exception occurred.
CPTEOTHR	Yes	An undefined exception occurred.

024040-200100

Usage Information

The LISTEN service lets user-written application programs implement TCP/IP server facilities. Server applications passively wait, then establish connections with single- or multi-thread support. The LISTEN service generalized parameter list (ACM) describes the application's communications requirements as well as information related to established connections. The ACM contains fields initialized by both a user application and by the LISTEN service, on completion.

There are required and optional fields initialized by a user or calling application. The ACM version number and the transport provider local port or service name are required. The selection of port or service name defines the server well-known port address. Optional fields control data transfer buffering, statistics, tracing, and subtask initialization.

When the LISTEN service completes or the data processing task executes, the ACM contains information related to the established connection. A token that identifies the connection is returned in the ACM, and must be used in all subsequent requests that refer to the connection. The application program should make no assumptions regarding the format of a token, other than it is an unsigned, full word value.

Information related to the negotiated buffer values, host names, host addresses, and transport provider addresses are returned in the ACM.

The version number (ACMVERS) indicates the CPT release level in which this user application program is written. This required field must be set to a binary 2 and is validated by the LISTEN service before processing the request.

The function code (ACMFUNC) indicates the CPT callable service ID. The field is initialized by the CPT service stub program. The function code identifies argument lists within the error or trace logs, and dump analysis.

The transport provider port number (ACMLPORT) or service name (ACMSRVCE) is required. These fields identify the well-known port to which a client initiates a connection request. The port number has precedence over service name. This implies that the service name field is only used if a port number is not specified.

The transaction ID field (ACMTRNID) identifies a data processing task. This is an optional field that causes the LISTEN service to execute continuously. The LISTEN service starts a new transaction after a client connection is established, then waits for additional connection requests. An updated ACM is passed to the data processing task. Control is not returned to the calling program until an error occurs. The return code indicates the reason for the failure. Errors indicating the transport provider, CICS, or CPT termination are acceptable. Errors indicating port in use, API unavailable, or program checks should be investigated.

User application programs can control CPT and transport provider data transfer buffering. The ACMQSEND, ACMMSEND, ACMQRECV, and ACMMRECV specify the number and size of buffers allocated. The SEND and RECEIVE buffers are allocated on initial entry into either the SEND or RECEIVE service. The corresponding values used by the SEND and RECEIVE services are independent of each other. The SEND service multiplies the queue and buffer values to determine output storage requirements. The RECEIVE service performs a

200801-024040-200100

similar function to determine input storage requirements. The product of the queue and buffer values cannot exceed 61,440. CPT requires some additional storage use to manage these buffers. This extra storage is included in the allocation.

The CPT SEND service uses the ACMQSEND value, and the CPT RECEIVE service uses the ACMQRECV value. These values indicate the maximum number of uncompleted SEND or RECEIVE requests that can be queued by the application to the transport provider. The CPT data transfer services schedule asynchronous transport provider or API requests on the caller's behalf. These API requests must be completed before they can be used again. If the queue values are too small the CPT data transfer service may block the caller's request and schedule a wait within the service routines. If the queue values are too large the user application might be wasting storage.

The CPT SEND service uses the ACMMSEND value, and the CPT RECEIVE service uses the ACMMRECV value. These values indicate the maximum number of user data bytes that can be transferred by the application in a single SEND or RECEIVE request to the transport provider. The user application is not limited to these values within the data transfer services. However, it is important to note that multiple transport provider or API requests are issued to complete the caller's request. Information on queue and buffer size is in the descriptions of **RCVFROM** on page 2-47 and **SEND** on page 2-67.

Initially, the tuning of data transfer storage may not be a concern. However, the ability to control storage allocation can prove beneficial to the application or CICS region. Queue size can increase data transfer throughput. You should consider enabling the statistics option to gather CPT statistical information. This information can set the SEND or RECEIVE queue and buffer size values.

The LISTEN service can modify the data transfer buffer allocation values. These values are negotiated with the transport provider and, depending on the site configuration, can be reduced. Any application dependent on these values should check them on return. These values are generally not modified when giving reasonable numbers. However, it is advisable to check with the site administrator for the maximum values of the API transport services.

A number of arguments are not set by the calling application, but are returned to the caller. These values represent information related to the client connection and can be used by the application. The local port, host name, and IP address are returned as well as the client's corresponding values. An ACM is passed the started transaction when a TRANSID is specified in the caller's listen argument list.

20-024040-200100

Client-Data Listener Option

The option ACMLTRAN is used in conjunction with ACMTIMEO and is mutually exclusive of the use of the ACMTRNID field. ACMLTRAN indicates to the LISTEN service that the connecting client application will specify what server functions to execute. When the LISTEN service receives a CONNECT request and ACMLTRAN is specified, it uses a partial record timed RECEIVE (see RECEIVE service options) to get the client's data. It uses ACMTIMEO to know how long to wait for the client data which can be in any of these formats:

```
TRAN
TRAN,UUUUUUUUUUUU
TRAN,UUUUUUUUUUUU,IC,HHMMSS
TDQN,UUUUUUUUUUUU,TD
TRAN,,IC,HHMMSS
TDQN,,TD
```

PARAMETER	DESCRIPTION
TRAN	A 1- to 4-character transaction ID to start, passing CLNTPARM to the started transaction
UUUUUUUUUUUU	A 1- to 35-byte user data and is passed to the started transaction or written to the transient data queue in the field CLNTDATA.
IC	Specifies that TRAN is to be started in HHMMSS; if left blank, startup is immediate.
HHMMSS	Hours, minutes, and seconds for IC option.
TD	Indicates that CLNTPARM will be written into the transient data queue, TDQN.

```
CLNTPARM  DS    OF
TOKEN     DS    F       New token
          DS    CL16    Reserved
CLNTDATA  DS    CL36    Up to 35 bytes of client data
PROTADDR  DS    OF
DOMAIN    DS    H       Family
RPORT     DS    H       Remote port from ACMRPORT
RADDR     DS    H       Remote IPADDR from ADMRADDR
          DS    XL8'00'Reserved
CLNTLEN   EQU   * - CLNTPARM
```

Note: Using this option puts a listener at risk of being tied up until the client actually sends the data. Set ACMTIMEO with this fact in mind. The new trace flag, ACMTCLTD, can optionally be specified to trace the CLNTPARM written to TDQN transient data queue via CPT918I.

200801-024040-200100

Example:

This example builds a simple server ACM and performs the LISTEN. The application program sets the argument list version to 1 and listens for connection requests on port 1234. Control is returned from the LISTEN service on establishment of a connection or by some error. Register 15 is tested to determine the success of the request. If register 15 is non-zero, an error has occurred and the diagnostic code indicates the reason for failure. If register 15 is zero, then the LISTEN service completed successfully and a token representing the data transfer connection is returned. The token is used for all CPT requests related to that connection. After a CPT CLOSE service is issued, the token is no longer valid and the application reissues the LISTEN service for additional connection requests.

This sample program is generally not the preferred server model. The problem is that after returning from the LISTEN service the application blocks additional incoming connection requests. A better example of this facility is shown in the next example. This single-threaded server model is really only suitable for connections of a very short time duration.

```
*         Dsect's
*
          T09DACM MF=DSECT    Argument for Connection Management
*
*         Working storage
*
DFHEISTG  DSECT
          .
          .
          .
LSTNARG   DS     XL(ACMLEN)   Argument for Connection Management
*
*         Entry
*
label  DFHEIENT
          .
          .
          .
*
*         CPT LISTEN Connection Management request
*
LISTEN    DS     0H
          LA     R2,LSTNARG   Load Argument for Connection Management
          USING ACM,R2
          MVC    ACMVERS,=H'2'        Set Version number
          MVC    ACMLPORT,=H'1234'    Set Server well-known port
          LR     R1,R2                Load ACM address in Reg 1
          L      R15,=V(T09FLSTN)     Load LISTEN service Stub address
          BALR   R14,R15              Issue LISTEN service
          LTR    R15,R15              Test Return Code
          BZ     LOADTOKN             Good, process data
          L      R4,ACMRTNCD          Load Return Code
          L      R5,ACMDGNCD          Load Diagnostic Code
          .
          . Process and log Connection Management request error
          .
          B      END        Termination Transaction
```

```
LOADTOKN  DS      0H
          L       R6,ACMTOKEN   Load (Data Transfer) Connection Token
          DROP    R2          ACM

          .
          . Application and CPT Data Transfer (SEND/RECEIVE) processing
          .

          .
          . CPT (Data Transfer) Connection Release
          .

          B       LISTEN        Listen for more Client connections
*
*         Terminate Transaction
*
END       DS      0H
          EXEC    CICS RETURN
```

Example:

This example builds a simple server ACM and performs the LISTEN request. The application program sets the argument list version to 1 and listens for connection requests on port 1234. Control is returned from the LISTEN service on establishment of a connection or by some error. Register 15 is tested to determine the success of the request. If register 15 is non-zero, an error has occurred and the diagnostic code indicates the reason for failure. If register 15 is zero, then the LISTEN service completed successfully and a token representing the data transfer connection is returned. The token is used for all CPT requests related to that connection. In this example, the token is passed to a new transaction and then the application reissues the LISTEN service for additional connection requests.

This sample program differs from the previous example in that data transfer is not performed by the listening transaction, but by a different transaction. This allows for a more efficient server program. The server application is better able to respond quickly to new connection requests, because it is not involved in the tedious task of data transfer or connection management after the initialization connection.

This example does not utilize the optional GIVE Facility Management service. The GIVE service is beneficial if this task was expected to terminate before the data processing transaction initiated. However, since this task is not expected to terminate any abnormal termination would release all CPT connections currently associated with this task.

```
*
*         Dsect's
*
          T09DACM MF=DSECT     Argument for Connection Management
*
*         Working storage
*
DFHEISTG  DSECT
          .
          .
          .
LSTNARG   DS     XL(ACMLEN)    Argument for Connection Management
*
*         Entry
*
label     DFHEIENT
          .
          .
          .
*
*         CPT LISTEN Connection Management request
*
          LA     R5,LSTNARG    Load Argument for Connection Management
          USING T09DACM,R5
          MVC    ACMVERS,=H'2'     Set Version number
          MVC    ACMLPORT,=H'1234'    Set Server well-known port
LISTEN    DS     0H
          LR     R1,R5             Load ACM address in Reg 1
          L      R15,=V(T09FLSTN) Load LISTEN service Stub address
          BALR   R14,R15           Issue LISTEN service
```

```
                LTR     R15,R15          Test Return Code
                BZ      START            Good, process data
                L       R3,ACMRTNCD      Load Return Code
                L       R4,ACMDGNCD      Load Diagnostic Code
                .
                . Process and log Connection Management request error
                .
                B       END              Termination Transaction
START           DS      0H

                EXEC CICS START TRANSID(trans-id)
                FROM(ACMTOKEN) LENGTH(4)

                B       LISTEN           Listen for more Client connections
                DROP    R5               ACM

*
*               Terminate Transaction
*
END             DS      0H
                EXEC    CICS RETURN
```

Example:

This example builds a server ACM and performs the LISTEN request. The application program listens for connection requests on whatever port is mapped into service name DISCARD. A TRANSID is specified and statistic flags are set. The T09MCALL assembler macro instruction sets the argument version number and calls the LISTEN service. Control is not returned from the LISTEN service until a failure has been detected. Register 15 is tested to determine the success of the request. If register 15 is non-zero, an error has occurred and the diagnostic code indicates the reason for failure. Errors indicating CPT or transport provider (API) termination are acceptable. The argument fields are not updated on return except for the return code (ACMRTNCD) and diagnostic code (ACMDGNCD).

This sample program differs from the previous example in that a START command for a new transaction is performed by the LISTEN service and not by the user application. Also, control is not returned to the calling application until a failure occurs. Generally, this failure is due to termination of CICS, CPT, or the transport provider (API).

In this example, transaction SRV3 is automatically started by the LISTEN service for each connection established on the DISCARD port.

```
*
*               Dsect's
*
                T09DACM MF=DSECTArgument for Connection Management
*
*               Working storage
*
DFHEISTG        DSECT
                .
                .
                .
LSTNARG         DS      XL(ACMLEN)    Argument for Connection Management
*
*               Entry
```

```
*label    DFHEIENT
          .
          .
          .
*
*         CPT LISTEN Connection Management request
*
          LA     R7,LSTNARG    Load Argument for Connection Management
          USING T09DACM,R7
          MVC    ACMSRVCE(7),=C'DISCARD' Set Server Service name
          MVC    ACMTRNID,=C'SRV3'   Set started trans id
          OI     ACMSTAT1,ACMSTCON+ACMSTFIN Set statistics flags
          T09MCALL LISTEN,PARM=LSTNARG  Issue Listen request
          LTR    R15,R15              Test Return Code
          BZ     END                  Good, Terminate transaction
          C      R15,=A(CPTEDRAN)     API drain RC?
          BE     END                  Equal, known RC
          C      R15,=A(CPTETERM)     API term RC?
          BE     END                  Equal, known RC
          C      R15,=A(CPTEPRGE)     API purge RC?
          BE     END                  Equal, known RC
          L      R3,ACMRTNCD          Load Return Code
          L      R4,ACMDGNCD          Load Diagnostic Code
          .
          . Process and log Connection Management request error
          .
          DROP   R5                   ACM
*
*         Terminate Transaction
*
END       DS     0H
          EXEC   CICS RETURN
          .
```

RCVFROM

The RCVFROM service lets you develop connectionless client and server applications. This service is UDP only. The RCVFROM service provides these basic functions:

- Establishes a UDP server endpoint represented by a new token and starts receiving datagrams on a user-specified well-known port. Indicate this function to the RCVFROM service by passing an ADTTOKEN equal to zero. RCVFROM then creates all the internal control blocks and the RCVFROM buffer queue. Even though the SENDTO buffer queue is not allocated for this endpoint (token) until the SENDTO service is called, the SENDTO buffer size and number must be specified at this time because they are negotiated with the transport provider and recorded in the internal CPT control blocks at endpoint creation time. Upon return from the RCVFROM service, ADTTOKEN contains the token value to be passed to subsequent RCVFROM and SENDTO service calls.
- Receives a datagram at a previously established UDP endpoint represented by an existing token. This functionality makes the RCVFROM service call just a data transfer call that can be used by a client or server application. The RCVFROM buffer queue is only allocated upon the first call to the RCVFROM service, whether or not ADTTOKEN is equal to zero.

UDP tokens created with the RCVFROM or SENDTO services cannot be passed to the TCP-only services, CONNECT, LISTEN, SEND, and RECEIVE. All other CPT service calls are available to UDP applications.

The non-blocking option of the RCVFROM service (ADTOPCD1=ADTNBLKR), allows applications to be developed that can poll a well-known UDP port or send to a remote UDP server and then make a predetermined number of RCVFROM calls to get back a response. Given the general unreliable nature of UDP, not blocking on a RCVFROM call can build in some flexibility with regards to handling lost datagrams.

This table describes the RCVFROM service arguments:

MACRO ID	DSECT NAME	SIZE	CREATED BY
T09DADT	ADT	644 (X'284')	User application

200801-024040-200100

Assembler Data Area

This is what the DSECT control block looks like in Assembler language:

```
Name      Operation Operands  Description

ADT       DSECT  ,
ADTVERS   DS     H       Version number
ADTFUNC   DS     H       Function code
ADTTOKEN  DS     A       Token (CEP)
ADTBUFFA  DS     A       Data buffer address
ADTBUFFL  DS     F       Data buffer length
ADTRTNCD  DS     F       Return code
```

```
ADTDGNCD  DS    F       Diagnostic code
ADTSTATS  DS    0F      Statistics flag
          DS    XL3
ADTSTAT   DS            Primary statistics request byte
ADTSTERM  EQU   X'02'    - Termination statistics

ADTTRACE  DS    0F      Trace flag
          DS    XL2
ADTTRAC2  DS    X       Second trace byte (for hi lvl lang)
ADTTTKNS  EQU   X'01'    - Trace token information
ADTTTPL   EQU   X'02'    - Trace TPL block
ADTTSTOR  EQU   X'08'    - Trace GETMAIN/FREEMAIN
ADTTRAC1  DS    X       First trace byte (for hi lvl lang)
ADTTNTRY  EQU   X'01'    - Trace entry points
ADTTARGS  EQU   X'02'    - Trace arguments
ADTTRECV  EQU   X'04'    - Trace TRECV
ADTTSEND  EQU   X'08'    - Trace TSEND
ADTTTERM  EQU   X'10'    - Trace termination
ADTTPASS  EQU   X'20'    - Trace TAKE
ADTTCLSE  EQU   X'40'    - Trace close
ADTTTERR  EQU   X'80'    - Trace TPL errors

ADTQSEND  DS    F       TSEND queue size
ADTMSEND  DS    F       Max TSEND TPL buffer size
ADTQRECV  DS    F       TRECV queue size
ADTMRECV  DS    F       Max TRECV TPL buffer size
ADTTIMEO  DS    F       Seconds to wait for timed RECEIVE
          DS    4F      Reserved
ADTLPORT  DS    H       Local transport provider port
ADTRPORT  DS    H       Remote transport provider port
          DS    H       Reserved
ADTSRVCE  DS    CL36    Local/remote service name
          DS    X       Reserved for C string
ADTSEP#   DS    X       Number of SEP chars-make fullwd
ADTSEP1   DS    X       First or only seperator character
ADTSEP2   DS    X       Second seperator character
          DS    H       Unused
ADTLADDR  DS    A       Local IP host address
ADTRADDR  DS    A       Remote IP host address
ADTLNAME  DS    CL255   Local IP host name
          DS    X       Reserved for C string
ADTRNAME  DS    CL255   Remote IP host name
          DS    X       Reserved for C string
ADTUCNTX  DS    F       User context field
ADTOPTNS  DS    0F      Data transfer option codes
ADTOPCD4  DS    X       Option code 4
ADTOPCD3  DS    X       Option Code 3
ADTOPCD2  DS    X       Option code 2
ADTFVLST  EQU   X'80'    Vector list flag
ADTNOSTP  EQU   X'40'    Do not strip LL or SEP SEQ on RECV
ADTOPCD1  DS    X       Option Code 1
ADTFDNR   EQU   X'80'    Do DNR name resol. For UDP, Def=No
ADTNBLKR  EQU   X'40'    Do not block on RECV/RECVFR (both)
ADTTMRCV  EQU   X'10'    Timed fullblk RECV w/ADTTIMEO
ADTTMPRT  EQU   X'08'    Timed partial RECV w/ADTTIMEO
ADTBLCKS  EQU   X'04'    Block on SEND (ICS)
ADTTYPLL  EQU   X'02'    LL type SEND/RECV
ADTTYPSP  EQU   X'01'    SEP type SEND/RECV
```

20-024040-200100

NAME	DESCRIPTION
ADTVERS	Version Indicates the CPT version number of the argument used by the calling program. This required field must be set to binary 2 for this release of CPT. Default: None
ADTFUNC	Function code Indicates the function or callable service ID requested by the application program this field should not be set by the application, but rather is initialized by the TRUE interface stub. Default: None
ADTTOKEN	Data transfer token Specifies a token that represents a UDP endpoint. If the ADT is being passed in a call to either the RCVFROM or SENDTO service, then the token can be zero, indicating to either service, to first create a token before sending or receiving a datagram. If the token is not zero, then it must be a token created previously by either the RCVFROM or SENDTO service. It is not necessary or efficient to create a token every time a CICS transaction calls the UDP data transfer services. It is an error to pass a TCP token to the UDP data transfer service routines, RCVFROM and SENDTO. Conversely it is an error to pass a UDP token to the TCP data transfer routines, RECEIVE and SEND. Default: 0
ADTBUFFA	User data address Indicates the storage address into which the UDP datagram is received (RCVFROM service). This is a contiguous segment of storage accessible to the user task. The content of all user data is application dependent, and not interpreted by either CPT nor the transport provider. The storage area can be aligned on any boundary convenient for the application program. Default: 0
ADTBUFFL	User data length Indicates the length (in bytes) of the buffer specified in ADTBUFFA which is to receive (RCVFROM service) the UDP datagram. If the incoming datagram does not fit into ADTBUFFA for a length of ADTBUFFL, then the warning, CPTWNEOM will be passed back to the caller in ADTRTNCD, indicating that more RCVFROM calls will be required to get the entire datagram. ADTBUFFL will indicate the actual length returned in ADTBUFFA. It is an error to call the RCVFROM service with an ADTBUFFL of zero.
ADTRTNCD	Return code Indicates the return code set by the RCVFROM service. Default: 0
ADTDGNCD	Diagnostic code Indicates the diagnostics code set by the RCVFROM service. This value generally indicates a transport provider return code. Default: 0

NAME	DESCRIPTION
ADTSTATS	Specifies logging options for the application program. The facility can be used for debugging and tuning during development. ◆ ACMSSTATS CONN– Specifies that messages be generated on the closing of a UDP token. These messages are generated by the CPT CLOSE service. The message numbers are CPT802Iand CPT803I. ◆ ACMSSTATS TERM– Specifies that messages be generated on terminating an established connection. These messages are generated by the CPT CLOSE service. The message numbers are CPT807I, CPT808I and CPT809I. Default: 0 (No statistics logging)
ADTTRACE	Specifies trace logging options for the application program. The facility can be used for debugging during development. ◆ ADTTNTRY – Specifies that a message be generated on entry to a CPT service routine. The message is generated by all CPT service routines. The message number associated with this option is CPT901I. ◆ ADTTARGS – Specifies that a hex dump of the caller's arguments be generated on entry and exit of a CPT service routine. Messages are generated by the corresponding service routine. The message numbers associated with this option are CPT902I, CPT903I, CPT904I, CPT911I, CPT912, CPT921I, CPT922I, CPT924I, CPT925I, CPT930I, and CPT932I. ◆ ADTTRECV – Specifies a hex dump of the transport provider's input (RECEIVE) data. Messages are generated by the CPT RCVFROM service. The message number associated with this option is CPT913I. ◆ ADTTSEND – Specifies a hex dump of the transport provider's output (SEND) data. Messages are generated by the CPT SENDTO service. The message number associated with this option is CPT914I. ◆ ADTTTERM – Specifies that a message be generated on termination of a CPT service routine. Messages are generated by all CPT service routines. The message number associated with this option is CPT908I. ◆ ADTTPASS – Specifies that a hex dump of resources related to a passed connection be logged. Messages are generated by the CPT GIVE and TAKE services. The message numbers associated with this option are CPT919I and CPT920I. ◆ ADTTCLSE – Specifies that a hex dump of resources related to a CLOSE processing of a connection be logged. Messages are generated by the CPT CLOSE service. The message number associated with this option is CPT906I ◆ ADTTTERR – Specifies that a hex dump be logged of a transport provider API parameter list that fails successful completion. The message number associated with this option is CPT400I. ◆ ADTTTKNS – Specifies that a hex dump of the UDP token be logged. Messages are generated by all service routines on entry. The message number associated with this option is CPT909I. ◆ ADTTTPL – Specifies that a hex dumped logged of the transport provider API parameter list. The message numbers associated with this option are CPT917I and CPT935I. ◆ ADTTSTOR – Specifies that a hex dump of storage management argument be logged. Messages are generated by various CPT service routines. The message numbers associated with this option are CPT928I and CPT929I. Default: 0 (No trace logging)

20-024040-200100

NAME	DESCRIPTION
ADTQSEND	API send queue size (used when ADTTOKEN=0) Specifies the maximum number of uncompleted SENDTO requests that can be queued by the application to the transport provider (API). This value lets applications control output processing and can affect throughput rates. The value is negotiated with the transport provider and can be modified by the transport provider. The total allocation for output processing is the product of the SENDTO queue and buffer size values and cannot exceed 61K. Default: 4
ADTMSEND	API send buffer size (used when ADTTOKEN=0) Specifies the maximum number of user data bytes that can be transferred by the application in a single SENDTO request to the transport provider (API). This value lets applications control output processing and can affect throughput rates. The value is negotiated with the transport provider and can be modified by the transport provider. The total allocation for output processing is the product of the SENDTO queue and buffer size values and cannot exceed 61K. Default: 4096
ADTQRECV	API RECEIVE queue size (used when ADTTOKEN=0) Specifies maximum number of uncompleted RCVFROM requests that can be queued by the application to the transport provider (API). This value lets applications control input processing and can affect throughput rates. The value is negotiated with the transport provider and can be modified by the transport provider. The total allocation for input processing is the product of the RCVFROM queue and buffer size values and cannot exceed 61K. Default: 4
ADTMRECV	API RECEIVE buffer size (used when ADTTOKEN=0) Specifies the maximum number of user data bytes that can be transferred by the application in a single RCVFROM request to the transport provider (API). This value lets applications control input processing and can affect throughput rates. The value is negotiated with the transport provider and can be modified by the transport provider. The total allocation for input processing is the product of the RCVFROM queue and buffer size values and cannot exceed 61K. Default: 4096
ADTTIMEO	RECEIVE time out value Not used by the RCVFROM service Default: 0
ADTLPORT	Local well-known service port (used when ADTTOKEN=0) Indicates the local transport layer port that the calling application will be receiving (RCVFROM) datagrams on. If the SENDTO service creates the token, this port number is assigned by the transport layer and returned to the caller. if the RCVFROM service creates the token, this is the well-known port requested by the caller. If the RCVFROM service is creating the token, this value or the transport layer service name (ADTSRVCE) must be specified. This field is an unsigned positive integer with a maximum value of 65,534. The value must be unique for each server application. Default: None
ADTRPORT	Remote port Indicates the remote transport layer port on which the incoming datagram originated. This value is returned to the caller of the RCVFROM service and may be different for each datagram received. This field is an unsigned positive integer with a maximum value of 65,534. Default: None

200801-024040-200100

NAME	DESCRIPTION
ADTSRVCE	Transport layer service name (used when ADTTOKEN=0) Indicates the local transport layer service name. This string is mapped into a transport layer address or port, through internal DNR. The resolved value is the well-known port to which the client application sends (SENDTO) UDP datagrams. This field has a maximum length of 36 bytes. The value must be unique for each server application. This field is optional and is not modified by the RCVFROM service. Default: None
ADTSEP#	Number of separator characters for option ADTTYPSP. Not used in the RCVFROM service. Default: None
ADTSEP1	First or only spaceport character for option ADTTYPSP. Not used in the RCVFROM service. Default: None
ADTSEP2	Second character or separator sequence for option ADTTYPSP Not used in the RCVFROM service. Default: None
ADTLADDR	Local IP host address Indicates the local host internet address. This field is an unsigned four-byte integer value. The local host internet address is returned to the caller of the RCVFROM Default: None
ADTRADDR	Remote IP host address Indicates the remote host internet address of the sender of the incoming UDP datagram. This value is returned to the caller of the RCVFROM service and may be different for each datagram received. This field is an unsigned four-byte integer value. Default: None
ADTLNAME	Local IP host name Indicates the local host internet name. This field is a 255-byte character string that is padded with blanks. The local host internet name is returned to the caller of the RCVFROM service. Default: None
ADTRNAME	Remote IP host name Indicates the remote host internet name. This field is a 255-byte character string that is padded with blanks. It is only resolved through internal DNR calls and returned to the caller of the UDP data transfer service routines (RCVFROM and SENDTO) if the ADTOPTNS flag, ADTFDNR is specified. This is to prevent the DNR call overhead on every UDP data transfer call. Default: None
ADTUCNTX	One word of user context Specifies one arbitrary word of user context to be associated with the endpoint. The information provided is not interpreted by CPT, and is merely saved with other endpoint information. Default: 0 (no user context)

NAME	DESCRIPTION
ADTOPTNS	Specifies data transfer options These are the ADT options that apply to UDP data transfer requests: ◆ ADTFDNR – Execute internal DNR calls during UDP data transfer service routine calls (RCVFROM and SENDTO) to resolve remote IP addresses into IP names in the that is padded with blanks. ◆ ADTNBLKR – Do not block on a call to the RCVFROM service. If no datagrams are currently available, the return code, CPTWBLCK, will be returned in ADTRTNCD. ◆ Default: None **Note:** These options can be toggled on every UDP data transfer call even if the caller is using the same token.

Network Considerations

The ADT is a common data structure used for both client and server UDP applications. There are common and unique values specified for a particular service request.

The table describes network considerations for Assembler API:

NAME	SERVER CONDITIONS FOR RCVFROM	CLIENT CONDITIONS FOR SENDTO
ADTLPORT	Local server well-known port selected by user application.	Local assigned transport provider port returned to user application.
ADTRPORT	Remote client transport provider port returned to user application.	Remote server transport provider well-known port selected by user application.
ADTSRVCE	Local server transport provider service name selected by user application.	Remote server transport provider service name selected by user application.
ADTRADDR	Remote IP host address returned to user application.	Remote IP host address selected by or returned to user application. The client must specify this field or ADTRNAME.
ADTLNAME	Local IP host name returned to user application.	Local IP host name Returned to user application.
ADTRNAME	Remote IP host name returned to user application only if ADTFDNR is specified in ADTOPCD1.	Remote IP host name selected by or returned to the user application. The client must specify this field or ADTRADDR. If ADTRADDR is used ADTRNAME will only be returned if ADTFDNR is specified in ADTOPCD1.

200801-024040-200100

Return Codes

The RCVFROM service returns a return code in registers 15 and 0 that indicate the results of the execution. These values are in the ADTRTNCD(R15) and ADTDGNCD (R0) within the argument for data transfer. The diagnostic code is optional and indicates the transport provider return code. Read **Appendix C – MESSAGES AND CODES** in the ***CICS Programmer's Toolkit Installation and Administration Guide*** for the return code cross reference table. This table describes the return codes for the RCVFROM service

RETURN	DIAGNOSTIC CODE	DESCRIPTION
CPTIRCOK		Successful
CPTEVERS		Control block version number is not supported.
CPTECONN	Yes	Requested host/service/port is not found.
CPTEPROT		RCVFROM called with a TCP token.
CPTEBUFF		Buffer address and/or length is invalid.
CPTENAPI	Yes	Transport provider API not available.
CPTETERM	Yes	Environment is being terminated.
CPTEPRGE	Yes	CPT interface terminating.
CPTEINTG	Yes	Transport provider integrity error.
CPTEENVR	Yes	Transport provider environment error
CPTEFRMT	Yes	Transport provider format error.
CPTEPROC	Yes	Transport provider procedure error.
CPTABEND		Abnormal exception occurred.
CPTEOTHR	Yes	An undefined exception occurred.

Examples

This example illustrates a simple connectionless server model. It accepts (RCVFROM) datagrams at a well-known port and echoes them back to their originator (SENDTO).

```
*-------------------------------------------------------------*
*
*   CICS DSECTS AND WORKING STORAGE SECTION
*
*-------------------------------------------------------------*
CPTPARMS  DS F                CPT CALLING PARAMETER
CPTADT    T09DADT MF=,PFX=ADT DATA TRANSFER ARGUMENT
CPTIOBUF  DS XL80             CPT SENDTO/RECVFR BUFFER
```

```
LABEL     DFHEIENT  CODEREG=(10)
    .
    .
    .
*-------------------------------------------------------------*
*
*        INDICATE TO RCVFROM THE WELL-KNOWN PORT FOR THIS
*        UDP SERVER
*
*-------------------------------------------------------------*
   MVC   ADTLPORT,=AL2(1113) UDP PORT TO SERVICE CLIENTS ON
*-------------------------------------------------------------*
*
*      HANG A RCVFROM ON THE ENDPOINT FOR THE PORT UNTIL A
*       DATAGRAM COMES IN
*
*-------------------------------------------------------------*
RECVIT    DS  0H
          LA  R03,CPTADT    LOAD DATA TRANSFER ARGUMENT ADDR
          ST  R03,CPTPARMS  SAVE ARGUMENT ADDRESS
          LA  R04,CPTIOBUF  GET THE ADDRESS OF OUR BUFFER
          ST  R04,ADTBUFFA  SET RCVFROM BUFFER ADDRESS
          LA  R05,L'CPTIOBUF GET MAX LENGTH OF DATAGRAM TO
                             RCVFROM
          ST  R05,ADTBUFFL  SAVE RCVFROM BUFFER LENGTH
*
      T09MCALL RCVFROM,PARM=CPTPARMS T09CRCFR ENTRY POINT
*
      LTR   R15,R15          GOOD RETURN CODE?
      BNE   EVALERR          NO, SOME OTHER ERROR
*-------------------------------------------------------------*
*
*      ECHO THE DATAGRAM BACK USING THE SAME ENPOINT (TOKEN)
*
*-------------------------------------------------------------*
*
*  BY USING THE SAME ADT, THE SENDTO CALL ALREADY HAS THE ADDRESS
*  AND LENGTH OF THE DATAGRAM JUST RECEIVED AS WELL AS WHERE TO
*  SEND IT BACK TO IN ADTRADDR AND ADTRPORT.  NOTE THAT THERE IS
*  NO NEED TO CREATE ANOTHER ENDPOINT (TOKEN) SINCE ALLOCATING
*  THE BUFFER QUEUES AND INTERNAL CPT CONTROL BLOCKS WOULD JUST
*  BE ADDED OVERHEAD.
*
      T09MCALL SENDTO,PARM=CPTPARMS T09CSNTO ENTRY POINT
*
      LTR   R15,R15       GOOD RETURN CODE?
      BZ    RECVIT        YES, GO WAIT FOR ANOTHER ONE
      B     EVALERR       NO, GO DIAGNOSE ERROR
```

200801-024040-200100

RECEIVE

The RECEIVE service receives data from a peer transport user connected to an endpoint. The RECEIVE service receives data as input on either a connection-mode (TCP) or connectionless-mode (UDP) transport service.

To invoke the RECEIVE service, a user application must first build an Argument for Data Transfer (ADT) and then issue a call to the RECEIVE routine. The ADT contains the version number, connection token, user buffer address, and length. When the RECEIVE service completes, the buffer length field is updated to reflect the amount of data processed by the RECEIVE service.

This table describes the RECEIVE service arguments:

MACRO ID	DSECT NAME	SIZE	CREATED BY
T09DADT	ADT	644 (X'284')	User application

Assembler Data Area

This is what the DSECT control block looks like in Assembler language:

```
Name      Operation Operands  Description

ADT       DSECT     ,
ADTVERS   DS        H         Version number
ADTFUNC   DS        H         Function code
ADTTOKEN  DS        A         Token (CEP)
ADTBUFFA  DS        A         Data buffer address
ADTBUFFL  DS        F         Data buffer length
ADTRTNCD  DS        F         Return code
ADTDGNCD  DS        F         Diagnostic code
ADTSTATS  DS        0F        Statistics flag
          DS        XL3
ADSTAT    DS        X         Priimary statistics request byte
ADTSTERM  EQU       X'02'      - Termination statistics
ADTTRACE  DS        0F        Trace flag
          DS        XL2
ADTTRAC2  DS        X         Second trace byte
                                 (for high level lang)
ADTTTKNS  EQU       X'01'      - Trace token information
ADTTTPL   EQU       X'02'      - Trace TPL block
ADTTSTOR  EQU       X'08'      - Trace getmain/freemain
ADTTRAC1  DS        X         First trace byte
                                 (for high level lang)
ADTTNTRY  EQU       X'01'      - Trace entry points
ADTTARGS  EQU       X'02'      - Trace arguments
ADTTRECV  EQU       X'04'      - Trace TRECV
ADTTSEND  EQU       X'08'      - Trace TSEND
ADTTTERM  EQU       X'10'      - Trace termination
ADTTPASS  EQU       X'20'      - Trace TAKE
ADTTCLSE  EQU       X'40'      - Trace CLOSE
ADTTTERR  EQU       X'80'      - Trace TPL errors

ADTQSEND  DS        F         TSEND queue size
ADTMSEND  DS        F         Maximum TSEND TPL buffer size
```

```
ADTQRECV  DS      F       TRECV queue size
ADTMRECV  DS      F       Maximum TRECV TPL buffer size
ADTTIMEO  DS      F       Seconds to wait for timed receive
          DS      4F      Reserved
ADTLPORT  DS      H       Local transport provider port
ADTRPORT  DS      H       Remote transport provider port
          DS      H       Reserved
ADTSRVCE  DS      CL36    Local/remote service name
          DS      X       Reserved for C string
ADTSEP#   DS      X       Number of SEP chars-make fullwd
ADTSEP1   DS      X       First or only seperator character
ADTSEP2   DS      X       Second seperator character
          DS      H       Unused
ADTLADDR  DS      A       Local IP host address
ADTRADDR  DS      A       Remote IP host address
ADTLNAME  DS      CL255   Local IP host name
          DS      X       Reserved for C string
ADTRNAME  DS      CL255   Remote IP host name
          DS      X       Reserved for C string
ADTUCNTX  DS      F       User context field
ADTOPTNS  DS      0F      Data transfer option codes
ADTOPCD4  DS      X       Option code 4
ADTOPCD3  DS      X       Option code 3
ADTOPCD2  DS      X       Option code 2
ADTFVLST  EQU     X'80'   Vector list flag
ADTNOSTP  EQU     X'40'   Do not strip LL or SEP SEQ on RECV
ADTOPCD1  DS      X       Option code 1
ADTFDNR   EQU     X'80'   Do DNR name resol. for UDP, Def=NO
ADTNBLKR  EQU     X'40'   Do not block on RECV/RECVFR (both)
ADTTMRCV  EQU     X'10'   Timed fullblk RECV w/ADTTIMEO
ADTTMPRT  EQU     X'08'   Timed partial RECV w/ADTTIMEO
ADTBLCKS  EQU     X'04'   Block on SEND (ICS)
ADTTYPLL  EQU     X'02'   LL type SEND/RECV
ADTTYPSP  EQU     X'01'   SEP type SEND/RECV
```

PARAMETER	DESCRIPTION
ADTVERS	Version Indicates the CPT version number of the argument used by the calling program. This required field must be set to a binary 2 for this release of CPT. Default: None
ADTFUNC	Function code Indicates the function or callable service ID requested by the application program. This field should not be set by the application, but is initialized by the TRUE interface stub program. Default: None

200801-024040-200100

PARAMETER	DESCRIPTION
ADTTOKEN	Data transfer token Specifies a token that represents a TCP connection. If the ADT is being passed in a call to either the RECEIVE or SEND service, then it must be a token representing a previously established TCP connection, via the CONNECT or LISTEN service. it is an error to pass a zero ADTTOKEN to either the RECEIVE or SEND service. It is an error to pass a TCP token to the UDP data transfer service routines, RCVFROM and SENDTO. Conversely, it is an error to pass a UDP token to the TCP data transfer routines. RECEIVE and SEND. Default: 0
ADTBUFFA	User data address Indicates the storage address into which network data is placed. This is a contiguous segment of storage accessible to the user task. The storage area can be aligned on any boundary convenient for the application program. Default: 0
ADTBUFFL	User data length Indicates the length (in bytes) of user data in the storage area as identified by the ADTBUFFA operand. The length is updated when the request is completed to reflect the actual length of user data received. This field must be interpreted on completion to determine the amount of data actually received. If a RECEIVE request is issued with a zero length, an error is detected and the request fails. Default: 0
ADTRTNCD	Return code Indicates the return code set by the RECEIVE service. This value is also returned in register 15 and indicates the success or failure of the service. Default: 0
ADTDGNCD	Diagnostic code Indicates the diagnostic code set by the service request. This value generally indicates a transport provider return code. Default: 0
ADTSTATS	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
ADTTRACE	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
ADTQSEND	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
ADTMSEND	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
ADTQRECV	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
ADTMRECV	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.

2040-A2 024040-200100

PARAMETER	DESCRIPTION
ADTTIMEO	RECEIVE timeout value Must be specified with these options: ◆ ADTTYPLL ◆ ADTTYPSP ◆ ADTTMRCV ◆ ADTTMPRT Specifying any of the above options on a RECEIVE call with an ADTTIMEO=ZERO will result in CPTETIME being returned in ADTRTNCD. Default: None
ADTLPORT	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
ADTRPORT	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
ADTSRVCE	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
ADTSEP#	Number of separator characters for option ADTTYPSP (0 < ADTSEP# < 3) If ADTSEP# is not equal to one or two, CPTESEP# will be returned in ADTRTNCD. Default: None
ADTSEP1	First or only separator character for option ADTTYPSP. Default: None
ADTSEP2	Second separator character in a sequence of two for option ADTTYPSP. Default: None
ADTLADDR	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
ADTRADDR	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
ADTLNAME	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
ADTRNAME	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
ADTUCNTX	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.

200801-024040-200100

PARAMETER	DESCRIPTION
ADTOPTNS	This field specifies data transfer options. These are the ADT options that apply to TCP data transfer requests: ◆ ADTNBLKR – Do not block on a call to the RECEIVE service If no data is currently available on the connection, CPTWBLCK will be returned in ADTRTNCD. ◆ ADTTMRCV – Timed full record RECEIVE These fields (along with other required ADT fields) are used to request a timed full record RECEIVE: ADTOPCD1 = ADTTMRCV ADTTIMEO > zero ADTBUFFL set to the length expected If the time limit expires before receiving any or all of the data specified by ADTBUFFL, CPTWTIMO will be returned in ADTRTNCD along with any data that was received. ◆ ADTTMPRT – Timed partial record RECEIVE These fields (along with other required ADT fields) are used to request a timed partial record RECEIVE: ADTOPCD1 = ADTTMPRT ADTTIMEO > zero ADTBUFFL set to maximum length expected If the time limit expires before receiving data, CPTWTIMO is returned in ADTRTNCD. If the time limit expires and any data is received, the data, along with a zero ADTRTNCD, will be returned to the caller. ◆ ADTTYPSP – SEP type RECEIVE These fields (along with other required ADT fields) are used to request a SEP type RECEIVE call: ADTOPCD1 = ADTTYPSP ADTSEP# = 1 or 2 ADTSEP1 = character ADTSEP2 = character if ADTSEP# = 2 ADTTIMEO > zero If the time limit expires and data is received, but no SEP characters are found, the data, along with an ADTRTNCD of CPTWNSEP will be returned to the caller. ◆ ADTTYPLL – LL type RECEIVE These fields (along with other required ADT fields) are used to request a SEP type RECEIVE call: ADTOPCD1 = ADTTYPLL ADTTIMEO > zero If the time limit expires before receiving any or all of the data specified by the LL (first two bytes of the data stream), CPTWTIMO will be returned in ADTRTNCD, along with any data that was received. ◆ ADTNOSTP – Do not strip record delimiter sequence This can be used with ADTTYPSP or ADTTYPLL to return the actual separator sequence or LL field in the buffer pointed to by ADTBUFFA. ◆ ADTBLCKS – Block on SEND service call (not used by RECEIVE service). ◆ ADTFVLST – Currently internal use only **Note:** It is an error to combine any of these RECEIVE service options: ADTNBLKR, ADTTYPLL, ADTTYPSP, ADTTMRCV, ADTTMPRT. An invalid combination causes CPTEOPTN to return in ADTRTNCD. Default: None

Completion Information

The RECEIVE service completes normally when the data is moved from the transport provider buffer to the application program's storage area. A length is returned to the application program, which is set to the amount of data actually processed.

Normal completion of the RECEIVE service implies that data has been moved to the user buffer. This does not necessarily indicate the application request was completely satisfied, but that some amount of data was processed. The user application is required to load the ADTBUFFL field to determine the actual data received. The RECEIVE service returns control to the calling application on receipt of a full buffer, a partial buffer, or an error indication. Control is returned to the user application with a partial buffer to avoid a WAIT command within the RECEIVE service. Additional requests to the RECEIVE service may be required to completely satisfy the user application's requirement.

The presence of exceptions or error conditions do not always indicate serious errors. A user application should check the return code to determine proper flow control. The release indication return code is an example of a condition that is not necessarily a serious error. This exception specifies that the remote host closed its half of the full-duplex data connection and will not send any additional data. This return code is acceptable, and generally indicates that graceful termination of the connection should begin.

On normal return to the application program, the general return code in register 15 (ADTRTNCD) is set to zero (CPTIRCOK). The diagnostic code in register 0 (ADTDGNCD) is always zero. The length field (ADTBUFFL) indicates the amount of data processed.

If the RECEIVE service completes abnormally, some or no user data may have been sent to the peer transport user. The general return code (ADTRTNCD) in register 15, and the diagnostic code (ADTDGNCD) in register 0, indicate the nature of the failure. The diagnostic code (ADTDGNCD) generally contains a specific code that is generated by the transport provider.

200801-024040-200100

Return Codes

The RECEIVE service returns a code in registers 15 and 0 that indicates the results of the execution. These values are in the ADTRTNCD (R15) and ADTDGNCD (R0) within the ADT. The diagnostic code is optional and generally indicates a transport provider return code. Read **Appendix C – MESSAGES AND CODES** in the ***CICS Programmer's Toolkit Installation and Administration Guide*** for the return code cross reference table.

This table describes the RECEIVE service return codes:

RETURN	DIAGNOSTIC CODE	DESCRIPTION
CPTIRCOK		Successful.
CPTEVERS		Control block version number is not supported.
CPTETOKN		Specified token is not valid.

RETURN	DIAGNOSTIC CODE	DESCRIPTION
CPTEBUFF		Buffer address and/or length is invalid.
CPTENAPI	Yes	Transport provider API is not available.
CPTETERM	Yes	Environment is being terminated.
CPTERLSE	Yes	Release indication.
CPTEDISC	Yes	Disconnect indication.
CPTEINTG	Yes	Transport provider API integrity error.
CPTEENVR	Yes	Transport provider API environment error.
CPTEFRMT	Yes	Transport provider API format error.
CPTEPROC	Yes	Transport provider API procedure error.
CPTABEND		Abnormal exception occurred.
CPTEOTHR	Yes	An undefined exception occurred.

Usage Information

The RECEIVE service receives normal data inputs through a CPT connection. The data may be part of a byte stream being received over a connection (TCP), or may be part of a datagram to be received via an association (UDP) to a peer transport user.

If the transport service type or protocol selected is a connection-mode byte stream (TCP), data is moved from the transport provider's storage area to the user application's storage area. Stream data may not be received with the same logical boundaries with which it was sent. However, the data arrives in the precise order in which it was sent. Possible fragmentation is a characteristic of stream data.

User applications may be required to issue multiple RECEIVE service requests to obtain all of the desired data. The data may arrive in particle segments. An application should be designed to handle such a situation. Additionally, users who write applications to process multiple record oriented data should consider including a mechanism to delimit the data. Design options can include a logical length field at the beginning of a record, or a special field, or fields, at the end. This lets the application determine record boundaries.

The RECEIVE service request is a synchronous operation, which may require the application to be blocked. The RECEIVE queue (ACMQRECV) and buffer (ACMMRECV) sizes, as specified during connection initialization, control input queuing and buffering. The queue size represents the number of uncompleted RECEIVE requests that can be pending to the transport provider. The default value is generally sufficient to support most applications. The transport provider normally completes a RECEIVE request on transferring data into the user buffer. However, data over the network may be fragmented and may require multiple requests to retrieve all of the data. The RECEIVE service issues a WAIT command if no data is available.

20 024040-200100

The queue and buffer size values are specified during connection initialization and can be modified by either the LISTEN or CONNECT services. An application that is dependent on these values should validate the requested values, compared with those values returned within the ACM. The values are modified if the transport provider site administrator has configured limits and the application request exceeds those values. If the requested values are modified, verify site definition statements for API transport services.

The version number (ADTVERS) indicates the CPT release level in which this user application program is written. This required field must be set to a binary 2 and is validated by the RECEIVE service before processing the request.

The function code (ADTFUNC) indicates the CPT callable service ID. The field is not initialized by a user application program and has little value to the application except for dump analysis. The function code identifies and maps an argument list with the error or trace log and dump analysis.

The token (ADTTOKEN) indicates the connection that is to receive data. This is a required field and is validated by the RECEIVE service before processing the request.

The data buffer address field ADTBUFFA is a full word. The application program assures that the residency mode of data areas it manages (for example, argument lists) is compatible with the addressing mode. The transport provider performs consistency checks on the addressing mode whenever a service request is issued. However, unpredictable results can occur before the transport provider can perform this check.

The data buffer length is indicated by the ADTBUFFL field. This is a full word positive integer. The data buffer length field should be less than or equal to the maximum receive buffer values. However, if the data buffer length is greater than the maximum receive buffer, the RECEIVE service attempts to satisfy the user's request with multiple transport provider requests. On return from the RECEIVE service, the ADTBUFFL is updated with a value that indicates the number of bytes processed.

The ADTOPTNS field specifies RECEIVE processing control options and provides a mechanism for event notification on return to the application program.

200P01-024040-200100

Example:

In this example, a message is received from a remote host. The token is loaded from the ACM. The ACM contains the version number, token, buffer address, and length. The return code is validated on return and the received data length is reloaded to determine the amount of data processed.

```
*
*         Dsect's
*
T09DADT MF=DSECT                    Argument for Data Transfer
*
*         Working storage
*
DFHEISTG  DSECT
          .
          .
          .
RECVARG   DS     XL(ADTLEN)         Data Transfer Argument
BUFFER    DS     CL256              Data Buffer
*
*         Entry
*
label     DFHEIENT
          .
          . CPT Connection Management initialization and request
          .
          L      R9,ACMTOKEN        Load ACM Token
          .
          . Application and CPT Data Transfer RECEIVE processing
          .
RECV      DS     H0
          LA     R8,RECVARG         Load RECEIVE argument address
          USING T09DADT,R8
          ST     R9,ADTTOKEN        Save connection token
          MVC    ADTVERS,=H'2'      Set version number
          LA     R3,BUFFER          Load address of output buffer
          ST     R3,ADTBUFFA        Save buffer address in DT arg
          L      R3,L'BUFFER        Load buffer length
          ST     R3,ADTBUFFL        Save buffer length in DT arg.
          LR     R1,R8              Load Data Transfer Arg. in R1
          L      R15,=V(T09FRECV)   Load RECEIVE Service Stub address
          BALR   R14,R15            Issue Receive request
          LTR    R15,R15            Test Return Code
          BNZ    ERROR              Non-zero, process error
          L      R3,ADTBUFFL        Load length of data received
          .
          . Application  processing
          .
          B      RECV               Receive more network data
*
*         RECEIVE Data Transfer Error
*
ERROR     DS     0H
          L      R3,ADTRTNCD        Load ADT Return Code
```

```
          C      R3,=A(CPTERLSE)  Test Release Indication
          BE     ORDER          Equal, graceful connection termination
          L      R4,ADTDGNCD      Load ADT Diagnostic Code
          .
          . Process and log application errors
          .
          B      ABORT            Abort Connection
          DROP   R8         ADT
*
*      CPT Connection Termination
*
ORDER     DS     0H
          .
          . Orderly Release of connection
          .
          B      END              Terminate Transaction
ABORT     DS     0H
          .
          . Abortive Release of connection
          .
*
*         Terminate Transaction
*
END       DS     0H
          EXEC   CICS RETURN
```

Example:

This example is similar to the first example, except the `T09MCALL` macro instruction adds the version number and generates the CPT `RECEIVE` stub routine call. The return code is validated on return and, if successful, processing continues.

```
*
*         Dsect's
*
          T09DADT MF=DSECT        Argument for Data Transfer
*
*         Working storage
*
DFHEISTG  DSECT
          .
          .
          .
RECVARG   DS     XL(ADTLEN)       Data Transfer Argument
BUFFER    DS     CL256            Data Buffer
*
*         Entry
*
label     DFHEIENT
          .
          . CPT Connection Management initialization and request
          .
          L      R9,ACMTOKEN      Load ACM Token
          .
          . Application and CPT Data Transfer RECEIVE processing
```

```
            .
RECV        DS     H0
            LA     R8,RECVARG        Load RECEIVE argument address
            USING T09DADT,R8
            ST     R9,ADTTOKEN       Save connection token
            LA     R3,BUFFER         Load address of output buffer
            ST     R3,ADTBUFFA       Save buffer address in DT arg
            L      R3,L'BUFFER       Load buffer length
            ST     R3,ADTBUFFL       Save buffer length in DT arg.
            T09MCALL RECEIVE,PARM=RECVARGIssue RECEIVE request
            LTR    R15,R15           Test Return Code
            BNZ    ERROR             Non-zero, process error
            L      R3,ADTBUFFL       Load length of data received

            .
            . Application  processing
            .

            B      RECV              Receive more network data
*
*       RECEIVE Data Transfer Error
*
ERROR       DS     0H
            L      R3,ADTRTNCD       Load ADT Return Code
            C      R3,=A(CPTERLSE)   Test Release Indication
            BE     ORDER       Equal, graceful connection termination
            L      R4,ADTDGNCD       Load ADT Diagnostic Code
            .
            . Process and log application errors
            .
            B   ABORT                Abort Connection

            DROP   R8                   ADT
*
*           CPT Connection Termination
*ORDER      DS     0H
            .
            . Orderly Release of connection
            .
            B      END               Terminate Transaction
ABORT       DS     0H
            .
            . Abortive Release of connection
            .
*
*       Terminate Transaction
*END        DS     0H
            EXEC   CICS RETURN
```

SEND

The SEND service sends data to a peer transport user connected to an endpoint. The SEND service sends data as output on either a connection-mode (TCP) or connectionless-mode (UDP) transport service.

To invoke the SEND service, a user application is required to first build an ADT and then to issue a call to the SEND routine. The ADT contains the version number, connection token, user buffer address, and length. When the SEND service completes, the buffer length field is updated to reflect the amount of data processed.

This table describes the SEND service arguments:

MACRO ID	DSECT NAME	SIZE	CREATED BY
T09DADT	ADT	644 (X'284')	User application

Assembler Data Area

This is what the DSECT control block looks like in Assembler language:

```
Name      Operation   Operands    Description

ADT       DSECT       ,
ADTVERS   DS          H           Version number
ADTFUNC   DS          H           Function code
ADTTOKEN  DS          A           Token(CEP)
ADTBUFFA  DS          A           Data buffer address
ADTBUFFL  DS          F           Data buffer length
ADTRTNCD  DS          F           Return code
ADTDGNCD  DS          F           Diagnostic code
ADTSTATS  DS          0F          Statistics flag
          DS          XL3
ADTSTAT   DS          X           Primary statistics request byte
ADTSTERM  EQU         X'02'        - Termination statistics

ADTTRACE  DS          0F          Trace flag
          DS          XL2
ADTTRAC2  DS          X           Second trace byte
*                                     (for high level language)
ADTTTKNS  EQU         X'01'        - Trace token information
ADTTTPL   EQU         X'02'        - Trace TPL block
ADTTSTOR  EQU         X'08'        _ Trace GETMAIN/FREEMAIN
ADTTRAC1  DS          X           First trace byte
*                                     (for high level language)
ADTTNTRY  EQU         X'01'        - Trace entry points
ADTTARGS  EQU         X'02'        - Trace arguments
ADTTRECV  EQU         X'04'        - Trace TRECV
ADTTSEND  EQU         X'08'        - Trace TSEND
ADTTPASS  EQU         X'20'        - Trace TAKE
ADTTCLSE  EQU         X'40'        - Trace CLOSE
ADTTTERR  EQU         X'80'        - Trace TPL errors

ADTQSEND  DS          F           TSEND queue size
ADTMSEND  DS          F           Max TSEND TPL buffer size
```

200801-024040-200100

```
ADTQRECV  DS        F       TRECV queue size
ADTMRECV  DS        F       Max TRECV TPL buffer size
ADTTTIMEO DS        F       Seconds to wait for timed
*                           RECEIVE
          DS        4F      Reserved
ADTLPORT  DS        H       Local transport provider
*                           port
ADTRPORT  DS        H       Remote transport provider
*                           port
          DS        H       Reserved
ADTSRVCE  DS        CL36    Local/remote service name
          DS        X       Reserved for C string
ADTSEP#   DS        X       Number of SEP chars
*                           make fullword
ADTSEP1   DS        X       First or only separator
*                           character
ADTSEP2   DS        X       Second separator character
          DS        H       Unused
ADTLADDR  DS        A       Local IP host address
ADTRADDR  DS        A       Remote IP host address
ADTLNAME  DS        CL255   Local IP host name
          DS        X       Reserved for C string
ADTRNAME  DS        CL255   Remote IP host name
          DS        X       Reserved for C string
ADTUCNTX  DS        F       User context field
ADTOPTNS  DS        OF      Data transfer option codes
ADTOPCD4  DS        X       Option code 4
ADTOPCD3  DS        X       Option code 3
ADTOPCD2  DS        X       Option code 2
ADTFVLST  EQU       X'8'    Vector list flag
ADTNOSTP  EQU       X'40'   Do not strip LL or SEP seq
*                           on RECV
ADTOPCD1  DS        X       Option 1
ADTFDNR   EQU       X'80'   Do DNR name resolution for
*                           UDP, Default=No
ADTNBLKR  EQU       X'40'   Do not block on
*                           RECV/RECVFR (both)
ADTTMRCV  EQU       X'10'   Timed fullblock RECV with
*                           ADTTIMEO
ADTTMPRT  EQU       X'08'   Timed partial RECV with
*                           ADDTIMEO
ADTBLCKS  EQU       X'04'   Block on send (ICS)
ADTTYPLL  EQU       X'02'   LL type SEND/RECV
ADTTYPSP  EQU       X'01'   SEP type SEND/RECV
```

PARAMETER	DESCRIPTION
ADTVERS	Version Indicates the CPT version number of the argument used by the calling program. This required field must be set to a binary 2 for this release of CPT. Default: None

2C 024040-200100

PARAMETER	DESCRIPTION
ADTFUNC	Function code Indicates the function or callable service ID requested by the application program. This field should not be set by the application, but rather is initialized by the TRUE interface stub program. Default: None
ADTTOKEN	Data transfer token Specifies a token that represents a TCP connection. If the ADT is being passed in a call to either the RECEIVE or SEND service, it must be a token representing a previously established TCP connection, via the CONNECT or LISTEN service. It is an error to pass a zero ADTTOKEN to either the RECEIVE or SEND service. It is an error to pass a TCP token to the UDP data transfer service routines, RCVFROM and SENDTO. Conversely, it is an error to pass a UDP token to the TCP data transfer routines, RECEIVE and SEND. Default: 0
ADTBUFFA	User data address Indicates the address of user data to be sent to the connected (or associated) transport user. This is a contiguous segment of storage accessible to the user task. The content of all user data is application-dependent, and is not interpreted by either CPT or the transport provider. The storage area can be aligned on any boundary convenient for the application program. Default: 0
ADTBUFFL	User data length Indicates the length (in bytes) of user data in the storage area as identified by the ADTBUFFA operand. The length is updated when the request is completed to reflect the actual length of user data sent. Generally, the length returned is equal to the length requested. If a SEND request is issued with a zero length, an error is detected and the request fails. Default: 0
ADTRTNCD	Return code Indicates the return code set by the SEND service. This value is also returned in register 15 and indicates the success or failure of the service. Default: 0
ADTDGNCD	Diagnostic code Indicates the diagnostic code set by the service request. This value generally indicates a transport provider return code. Default: 0
ADTSTATS	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
ADTTRACE	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
ADTQSEND	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.

200801-024040-200100

PARAMETER	DESCRIPTION
ADTMSEND	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
ADTQRECV	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
ADTMRECV	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
ADTTIMEO	RECEIVE timeout value Not used by the SEND service. Default: None
ADTLPORT	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
ADTRPORT	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
ADTSRVCE	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
ADTSEP#	Number of separator characters for option ADTTYPSP (0 < ADTSEP# < 3). If ADTSEP# is not equal to 1 or 2, CPTESEP# will be returned in ADTRTNCD. Default: None
ADTSEP1	First or only separator character for option ADTTYPSP. Default: None
ADTSEP2	Second separator character in a sequence of two for option ADTTYPSP. Default: None
ADTLADDR	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
ADTRADDR	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
ADTLNAME	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
ADTRNAME	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
ADTUCNTX	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.

20-024040-200100

PARAMETER	DESCRIPTION

ADTOPTNS	This field specifies data transfer options.
	
	These are the ADT options that apply to TCP data transfer requests:
	
	◆ ADTNBLKR – Do not block on a call to the RECEIVE service. Not used by the SEND service.
	
	◆ ADTTMRCV – Timed full record RECEIVE. Not used by the SEND service.
	
	◆ ADTTMPRT – Timed partial record RECEIVE. Not used by the SEND service.
	
	◆ ADTTYPSP – SEP type SEND. These files (along with other required ADT fields) are used to request a SEP type SEND call:
	
	`ADTOPCD1 = ADTTYPSP`
	`ADTSEP# = 1 OR 2`
	`ADTSEP1 = character`
	`ADTSEP2 = character if ADTSEP# = 2`
	
	◆ ADTTYPLL – LL type SEND. These fields (along with other required ADT fields) are used to request a SEP type SEND call:
	
	`ADTOPCD1 = ADTTYPLL`
	
	◆ ADTBLCKS – Block on SEND service call. The default for the SEND service is to return control to the caller as soon as the SEND request has been scheduled with the local transport provider. Option ADTBLCKS will block the return to the caller until the SEND data has been moved to the local transport provider's address space (not the remote TCP). This can provide TCP connection status at the SEND service call time rather than at some later time (next SEND or RECEIVE call).
	
	◆ ADTFVLST – Currently for internal use only.
	
	Note: It is an error to combine any of these SEND service options:
	
	`ADTTYPLL`
	`ADTTYPSP`
	
	An invalid combination will result in CPTEOPTN being returned in ADTRTNCD.
	
	Default: None

200801-024040-200100

Completion Information

The SEND service completes normally when the data has been both moved from the application program's storage area and forwarded to the transport provider for sending to the connected (or associated) transport user. A length is returned to the application program, which is set to the amount of data processed.

Normal completion of the SEND service implies nothing in regard to when the data is sent to the peer transport user. This only means that the transport provider has taken custody of the user data and the storage area provided by the application program can be reused. The transport provider generally sends the buffered data, but this may not occur synchronously with the completion of the SEND service.

On normal return to the application program, the general return code in register 15 (ADTRTNCD) is set to zero (CPTIRCOK). The diagnostic code in register 0 (ADTDGNCD) is always zero. The length field (ADTBUFFL) indicates the amount of data processed.

If the SEND service completes abnormally, some or no user data may have been sent to the peer transport user. The general return code (ADTRTNCD) in register 15, and the diagnostic code (ADTDGNCD) in register 0, indicate the nature of the failure. The diagnostic code (ADTDGNCD) generally contains a specific code that is generated by the transport provider.

Return Codes

The SEND service returns a code in registers 15 and 0 that indicates the results of the execution. These values are in the ADTRTNCD (R15) and ADTDGNCD (R0) within the ADT. The diagnostic code is optional and generally indicates a transport provider return code. Read **Appendix C – MESSAGES AND CODES** in the ***CICS Programmer's Toolkit Installation and Administration Guide*** for the return code cross reference table.

This table describes the SEND service return codes:

RETURN	DIAGNOSTIC CODE	DESCRIPTION
CPTIRCOK		Successful.
CPTEVERS		Control block version number is not supported.
CPTETOKN		Specified token is not valid.
CPTEBUFF		Buffer address and/or length is invalid.
CPTEPRGE	Yes	CPT Interface is terminating.
CPTENAPI	Yes	Transport provider API is not available.
CPTETERM	Yes	Environment is being terminated.
CPTERLSE	Yes	Release indication.
CPTEDISC	Yes	Disconnect indication.

RETURN	DIAGNOSTIC CODE	DESCRIPTION
CPTEINTG	Yes	Transport provider API integrity error.
CPTEENVR	Yes	Transport provider API environment error.
CPTEFRMT	Yes	Transport provider API format error.
CPTEPROC	Yes	Transport provider API procedure error.
CPTABEND		Abnormal exception occurred.
CPTEOTHR	Yes	An undefined exception occurred.

Usage Information

The SEND service sends normal data as output through a CPT connection. The data may be part of a byte stream being sent over a connection (TCP), or may be part of a datagram to be sent via an association (UDP) to a peer transport user.

If the transport service type or protocol selected is a connection-mode byte stream (TCP), data is moved from the application program's storage area to storage areas maintained by the transport provider. The data is packetized and sent to the connection transport user. Logical boundaries are not preserved in the data stream. The data is delivered to the peer transport user in the precise order in which it was sent. However, this data may be fragmented.

Data is not necessarily packetized and sent by the transport provider each time a SEND service is issued, nor is it sent when a buffer boundary is indicated. The transport provider may intentionally delay sending data as the result of performance optimization or congestion avoidance algorithms. Normally, data generated by the application is forwarded when it is sent in a continuous flow.

The SEND service request is a synchronous operation, which may require the application to be blocked. The SEND queue (ACMQSEND) and buffer (ACMMSEND) sizes, as specified during connection initialization, control output queuing and buffering. The queue size represents the number of uncompleted SEND requests that can be pending to the transport provider. The default value is generally sufficient to support most applications. The transport provider normally completes a SEND request within a reasonable amount of time, but a congested network may cause the transport provider delays in completing requests, and cause the application to be blocked.

Additionally, for an application that issues multiple SEND requests contiguously, consider increasing the default queue size. The buffer size represents the maximum number of user data bytes that can be transferred by the application in a single SEND request to the transport provider. This value is application dependent. A small value causes the SEND service to issue multiple transport provider SEND requests. Multiple transport provider SEND requests do not present a problem, but do reduce the queue size and can cause the application to be blocked. A large buffer value can waste application storage.

The queue and buffer size values are specified during connection initialization and can be modified on return. An application that is dependent on these values should validate the requested values, compared with values returned within the ACM. The values are modified if the transport provider site

200801-024040-200100

administrator has configured limits and the application request exceeds those values. If the requested values are modified, verify site definition statements for API transport services.

The version number (ADTVERS) indicates the CPT release level in which this user application program is written. This required field must be set to a binary 2 and is validated by the SEND service before processing the request.

The function code (ADTFUNC) indicates the CPT callable service ID. The field is not initialized by a user application program and has little value to the application except for dump analysis. The function code identifies and maps an argument list with the error or trace log and dump analysis.

The token (ADTTOKEN) indicates the connection that is to transmit data. This required field is validated by the SEND service before processing the request.

The data buffer address field ADTBUFFA is a full word. The application program assures that the residency mode of data areas it manages (for example, argument lists) is compatible with the addressing mode. The transport provider performs consistency checks on the addressing mode whenever a service request is issued. However, unpredictable results may occur before the transport provider can perform this check.

The data buffer length is indicated by the ADTBUFFL field. This is a full word positive integer. The data buffer length field should be less than or equal to the maximum send buffer values. However, if the data buffer length is greater than the maximum send buffer, the SEND service fragments the user data into multiple transport provider requests. The ADTBUFFL is updated on return from the SEND service with a value that indicates the number of bytes processed.

The ADTOPTCD field specifies SEND processing control options and provides a mechanism for event notification on return to the application program. Currently, this facility is reserved for internal processing.

Example:

In this example, a welcome message is sent to the remote connection. The token is loaded from the ACM. The ADT contains the version number, token, buffer address and length. The return code is validated on return and, if successful, processing continues.

```
*
*         Dsect's
*
          T09DADT MF=DSECT        Argument for Data Transfer
*
*         Working storage
*
DFHEISTG  DSECT
          .
          .
          .
SENDARG   DS      XL(ADTLEN)      Data Transfer Argument
WELCOME   DC      C'Welcome to the wonderful world of CPT/API.'
*
*         Entry
*
label     DFHEIENT
```

```
                .
                . CPT Connection Management initialization and request
                .
                L       R9,ACMTOKEN       Load ACM Token

                .
                . Application and CPT Data Transfer SEND processing
                .
SEND            DS      H0
                LA      R7,SENDARG        Load SEND argument address
                USING T09DADT,R7
                ST      R9,ADTTOKEN       Save connection token
                MVC     ADTVERS,=H'2'     Set version number
                LA      R3,WELCOME        Load address of output buffer
                ST      R3,ADTBUFFA       Save buffer address in DT arg
                L       R3,L'WELCOME      Load buffer length
                ST      R3,ADTBUFFL       Save buffer length in DT arg.
                LR      R1,R7             Load Data Transfer Arg. in R1
                L       R15,=V(T09FSEND) Load SEND Service Stub address
                BALR    R14,R15           Issue SEND request
                LTR     R15,R15           Test Return Code
                BNZ     ERROR             Non-zero, process error

                .
                . Application  processing
                .
                B       SEND              Send more network data
*
*       SEND Data Transfer Error
*
ERROR           DS      0H
                L       R3,ADTRTNCD       Load ADT Return Code
                C       R3,=A(CPTERLSE)   Test Release Indication
                BE      ORDER          Equal, graceful connection termination
                L       R4,ADTDGNCD       Load ADT Diagnostic Code
                .
                . Process and log application errors
                .
                B       ABORT             Abort Connection

                DROP    R7                ADT
*
*           CPT Connection Termination
*
ORDER           DS      0H
                .
                . Orderly Release of connection
                .
                B       END               Terminate Transaction
ABORT           DS      0H
                .
                . Abortive Release of connection
                .
*
*           Terminate Transaction
*
END             DS      0H
                EXEC    CICS RETURN
```

200801-024040-200100

Example:

This example is similar to the first example, except the `T09MCALL` macro instruction adds the version number and generates the CPT `SEND` stub routine call. The return code is validated on return and, if successful, processing continues.

```
*
*          Dsect's
*
           T09DADT MF=DSECTArgument for Data Transfer
*
*          Working storage
*
DFHEISTG   DSECT
           .
           .
           .
SENDARG    DS     XL(ADTLEN)       Data Transfer Argument
WELCOME    DC     C'Welcome to the wonderful world of CPT/API.'
*
*          Entry
*
label      DFHEIENT
           .
           . CPT Connection Management initialization and request
           .
           L      R9,ACMTOKEN      Load ACM Token
           .
           . Application and CPT Data Transfer SEND processing
           .
SEND       DS     H0
           LA     R2,SENDARG       Load SEND argument address
           USING T09DADT,R2
           ST     R9,ADTTOKEN      Save connection token
           MVC    ADTVERS,=H'2'    Set version number
           LA     R3,WELCOME       Load address of output bufferd
           ST     R3,ADTBUFFA      Save buffer address in DT arg
           L      R3,L'WELCOME     Load buffer length
           ST     R3,ADTBUFFL      Save buffer length in DT arg.
           T09MCALL SEND,PARM=SENDARG Issue SEND request
           LTR    R15,R15          Test Return Code
           BNZ    ERROR            Non-zero, process error
           .
           . Application  processing
           .
           B      SEND             Send more network data
*
*          SEND Data Transfer Error
*
ERROR      DS     0H
           L      R3,ADTRTNCD      Load ADT Return Code
           C      R3,=A(CPTERLSE)  Test Release Indication
           BE     ORDER         Equal, graceful connection termination
           L      R4,ADTDGNCD      Load ADT Diagnostic Code
```

SC24040-200100

```
                .
                . Process and log application errors
                .
                B       ABORT           Abort Connection
                DROP    R2              ADT
*
*               CPT Connection Termination
*
ORDER           DS      0H
                .
                . Orderly Release of connection
                .
                B       END             Terminate Transaction
ABORT           DS      0H
                .
                . Abortive Release of connection
                .
*
*               Terminate Transaction
*
END             DS      0H
                EXEC    CICS RETURN
```

200801-024040-200100

SENDTO

This service is provided to allow connectionless client and server applications to be developed. This service is UDP only. The SENDTO service provides two basic functions:

- Establish a UDP client endpoint represented by a new token and send a datagram to a remote UDP server. This function is indicated to the SENDTO service by passing an ADTTOKEN equal to zero. SENDTO will then create all the internal control blocks and the SENDTO buffer queue. Even though the RCVFROM buffer queue will not be allocated for this endpoint (token) until the RCVFROM service is called, the RCVFROM buffer size and number must be specified at this time because they are negotiated with the transport provider and recorded in the internal CPT control blocks at endpoint creation time. Upon return from the SENDTO service ADTTOKEN will contain the token value to be passed to subsequent SENDTO and RCVFROM service calls.
- Send a datagram at a previously established UDP endpoint represented by an existing token. This functionality makes the SENDTO service call just a data transfer call that can be used by a client or server application. The SENDTO buffer queue is only allocated upon the first call to the SENDTO service whether ADTTOKEN is equal to zero or not.

UDP tokens created with the RCVFROM or SENDTO services cannot be passed to the TCP only services, CONNECT, LISTEN, SEND, and RECEIVE. All other CPT service calls are available to UDP applications.

This table describes the arguments for the SENDTO service:

MACRO ID	DSECT NAME	SIZE	CREATED BY
T09DADT	ADT	644 (X'284')	User application

Assembler Data Area

This is what the DSECT control block looks like in Assembler language:

```
Name      Operation Operands  Description

ADT       DSECT     ,
ADTVERS   DS        H         Version number
ADTFUNC   DS        H         Function code
ADTTOKEN  DS        A         Token (CEP)
ADTBUFFA  DS        A         Data buffer address
ADTBUFFL  DS        F         Data buffer length
ADTRTNCD  DS        F         Return code
ADTDGNCD  DS        F         Diagnostic code
ADTSTATS  DS        0F        Statistics flag
          DS        XL3
ADTSTAT   DS        X         Primary statistics request byte
ADTSTERM  EQU       X'02'      - Termination statistics

ADTTRACE  DS        0F        Trace flag
          DS        XL2
```

```
ADTTRAC2  DS      X         Second trace byte (for hi lvl
*                           lang)
ADTTTKNS  EQU     X'01'      - Trace token information
ADTTTPL   EQU     X'02'      - Trace TPL block
ADTTSTOR  EQU     X'08'      - Trace GETMAIN/FREEMAIN
ADTTRAC1  DS      X         First trace byte (for hi lvl
*                           lang)
ADTTNTRY  EQU     X'01'      - Trace entry points
ADTTARGS  EQU     X'02'      - Trace arguments
ADTTRECV  EQU     X'04'      - Trace TRECV
ADTTSEND  EQU     X'08'      - Trace TSEND
ADTTTERM  EQU     X'10'      - Trace termination
ADTTPASS  EQU     X'20'     - Trace TAKE
ADTTCLSE  EQU     X'40'     - Trace CLOSE
ADTTTERR  EQU     X'80'     - Trace TPL errors

ADTQSEND  DS      F         TSEND queue size
ADTMSEND  DS      F         Max TSEND TPL buffer size
ADTQRECV  DS      F         TRECV queue size
ADTMRECV  DS      F         Max TRECV TPL buffer size
ADTTIMEO  DS      F         Seconds to wait for timed RECEIVE
          DS      4F        RESERVED
ADTLPORT  DS      H         Local transport provider port
ADTRPORT  DS      H         Remote transport provider port
          DS      H         Reserved
ADTSRVCE  DS      CL36      Local/remote service name
          DS      X         Reserved for C string
ADTSEP#   DS      X         Number of SEP chars-make fullwd
ADTSEP1   DS      X         First or only seperator character
ADTSEP2   DS      X         Second seperator character
          DS      H         Unused
ADTLADDR  DS      A         Local IP host address
ADTRADDR  DS      A         Remote IP host address
ADTLNAME  DS      CL255     Local IP host name
          DS      X         Reserved for C string
ADTRNAME  DS      CL255     Remote IP host name
          DS      X         Reserved for C string
ADTUCNTX  DS      F         User context field
ADTOPTNS  DS      0F        Data transfer option codes
ADTOPCD4  DS      X         Option code 4
ADTOPCD3  DS      X         Option code 3
ADTOPCD2  DS      X         Option code 2
ADTFVLST  EQU     X'80'     Vector list flag
ADTNOSTP  EQU     X'40'     Do not strip LL or SEP seq on RECV
ADTOPCD1  DS      X         Option code 1
ADTFDNR   EQU     X'80'     Do DNR name resol. for UDP, Def=No
ADTNBLKR  EQU     X'40'     Do not block on RECV/RECVFR (both)
ADTTMRCV  EQU     X'10'     Timed fullblk RECV w/ADTTIMEO
ADTTMPRT  EQU     X'08'     Timed partial RECV w/ADTTIMEO
ADTBLCKS  EQU     X'04'     Block on SEND (ICS)
ADTTYPLL  EQU     X'02'     LL type SEND/RECV
ADTTYPSP  EQU     X'01'     SEP type SEND/RECV
```

200801-024040-200100

NAME	DESCRIPTION
ADTVERS	Version Indicates the CPT version number of the argument used by the calling program. This required field must be set to binary 2 for this release of CPT. Default: None
ADTFUNC	Function code Indicates the function or callable service ID requested by the application program. This field should not be set by the application, but rather is initialized by the true interface stub. Default: None
ADTTOKEN	Data transfer token Specifies a token that represents a UDP endpoint. If the ADT is being passed in a call to either the RCVFROM or SENDTO service, the token can be zero, indicating to either service, to first create a token before sending or receiving a datagram. If the token is not zero, it must be a token created previously by either the RCVFROM or SENDTO service. It is not necessary or efficient to create a token every time a CICS transaction calls the UDP data transfer services. It is an error to pass a TCP token to the UDP data transfer service routines, RCVFROM and SENDTO. Conversely, it is an error to pass a UDP token to the TCP data transfer routines, RECEIVE and SEND. Default: 0
ADTBUFFA	User data address Indicates the storage address from which the UDP datagram will be sent (SENDTO service). This is a contiguous segment of storage accessible to the user task. The content of all user data is application dependent, and not interpreted by either CPT nor the transport provider. The storage area can be aligned on any boundary convenient for the application program. Default: 0
ADTBUFFL	User data length Indicates the length (in bytes) of the buffer specified in ADTBUFFA which is to be sent (SENDTO service). It is an error to call the SENDTO service with an ADTBUFFL of zero. Upon return to the caller, ADTBUFFL reflects the number of bytes actually sent (generally the number requested). Indicates the return code set by the SENDTO service. Default: 0
ADTRTNCD	Return code Indicates the return code set by the SENDTO service. This value is also returned in register 15 and indicates the success or failure of the service. Default: 0
ADTDGNCD	Diagnostic code Indicates the diagnostics code set by the SENDTO service. This value generally indicates a transport provider return code. Default: 0
ADTSTATS	ADTSTATS CONN | ADTSTSTS TERM Specifies logging options for the application program. The facility can be used for debugging and tuning during development. ◆ ADTSTATS CONN– Specifies that messages be generated on the closing of a UDP token. These messages are generated by the CPT CLOSE service. The message numbers are CPT802Iand CPT803I. ◆ ADTSTATS TERM– Specifies that messages be generated on terminating an established connection. These messages are generated by the CPT CLOSE service. The message numbers are CPT807I, CPT808I and CPT809I. Default: 0 (No statistics logging)

20-024040-200100

NAME	DESCRIPTION
ADTTRACE	This field specifies trace logging options for the application program. The facility can be used for debugging during development. ◆ ADTTNTRY – Specifies that a message be generated on entry to a CPT service routine. The message is generated by all CPT service routines. The message number associated with this option is CPT901I. ◆ ADTTARGS – Specifies that a hex dump of the caller's arguments be generated on entry and exit of a CPT service routine. Messages are generated by the corresponding service routine. The message numbers associated with this option are CPT902I, CPT903I, CPT904I, CPT911I, CPT912, CPT921I, CPT922I, CPT924I, CPT925I, CPT930I, and CPT932I. ◆ ADTTRECV – Specifies that a hex dump of the transport provider's input (RECEIVE) data. Messages are generated by the CPT RCVFROM service. The message number associated with this option is CPT913I. ◆ ADTTSEND – Specifies that a hex dump of the transport provider's output (SEND) data. Messages are generated by the CPT SENDTO service. The message number associated with this option is CPT914I. ◆ ADTTTERM – Specifies that a message be generated on termination of a CPT service routine. Messages are generated by all CPT service routines. The message number associated with this option is CPT908I. ◆ ADTTPASS – Specifies that a hex dump of resources related to a passed connection be logged. Messages are generated by the CPT GIVE and TAKE services. The message numbers associated with this option are CPT919I and CPT920I. ◆ ADTTCLSE – Specifies that a hex dump of resources related to a CLOSE processing of a connection be logged. Messages are generated by the CPT CLOSE service. The message number associated with this option is CPT906I ◆ ADTTTERR – Specifies that a hex dump be logged of a transport provider API parameter list that fails successful completion. The message number associated with this option is CPT400I. ◆ ADTTTKNS – Specifies that a hex dump of the UDP token be logged. Messages are generated by all service routines on entry. The message number associated with this option is CPT909I. ◆ ADTTTPL – Specifies that a hex dump of the transport provider API parameter list be logged. The message numbers associated with this option are CPT917I and CPT935I. ◆ ADTTSTOR – Specifies that a hex dump of storage management argument be logged. Messages are generated by various CPT service routines. The message numbers associated with this option are CPT928I and CPT929I. Default: 0 (No trace logging)
ADTQSEND	API SEND queue size (used when ADTTOKEN=0) Specifies the maximum number of uncompleted SENDTO requests that can be queued by the application to the transport provider (API). This value lets applications control output processing and can affect throughput rates. The value is negotiated with the transport provider and can be modified by the transport provider. The total allocation for output processing is the product of the SENDTO queue and buffer size values and cannot exceed 61K. Default: 4
ADTMSEND	API send buffer size (used when ADTTOKEN=0) Specifies the maximum number of user data bytes that can be transferred by the application in a single SENDTO request to the transport provider (API). This value lets applications control output processing and can affect throughput rates. The value is negotiated with the transport provider and can be modified by the transport provider. The total allocation for output processing is the product of the SENDTO queue and buffer size values and cannot exceed 61K. Default: 4096

200801-024040-200100

NAME	DESCRIPTION
ADTQRECV	API RECEIVE queue size (used when ADTTOKEN=0) Specifies maximum number of uncompleted RCVFROM requests that can be queued by the application to the transport provider (API). This value lets applications control input processing and can affect throughput rates. The value is negotiated with the transport provider and can be modified by the transport provider. The total allocation for input processing is the product of the RCVFROM queue and buffer size values and cannot exceed 61K. Default: 4
ADTMRECV	API RECEIVE buffer size (used when ADTTOKEN=0) Specifies the maximum number of user data bytes that can be transferred by the application in a single RCVFROM request to the transport provider (API). This value lets applications control input processing and can affect throughput rates. The value is negotiated with the transport provider and can be modified by the transport provider. The total allocation for input processing is the product of the RCVFROM queue and buffer size values and cannot exceed 61K. Default: 4096
ADTTIMEO	RECEIVE timeout value Not used by the SENDTO service Default: 0
ADTLPORT	Local well-known service port. Indicates the local transport layer port that the calling application will be sending (SENDTO) UDP datagrams from. If the SENDTO service creates the token, this port number is assigned by the transport layer and returned to the caller. If the RCVFROM service creates the token, this is the well-known port requested by the caller. This field is an unsigned positive integer with a maximum value of 65,534. The value must be unique for each server application. Default: None
ADTRPORT	Remote port. Indicates the remote transport layer port destination for the datagram being processed by the SENDTO service. This field is an unsigned positive integer with a maximum value of 65,534. Default: None
ADTSRVCE	Transport layer service name Not used in the SENDTO. Default: None
ADTSEP#	Number of separator characters for option ADTTYPSP. Not used in the SENDTO service. Default: None
ADTSEP1	First or only separator character for option ADTTYPSP Not used in the SENDTO service. Default: None
ADTSEP2	Second character or separator sequence for option ADTTYPSP. Not used in the SENDTO service. Default: None
ADTLADDR	Local IP host address Indicates the local host internet address. this field is an unsigned four-byte integer value. The local host internet address is returned to the caller of the SENDTO service. Default: None

NAME	DESCRIPTION
ADTRADDR	Remote IP host address Indicates the remote host internet address destination for the datagram being processed by the SENDTO service. This field is a unsigned four-byte integer value. Default: None
ADTLNAME	Local IP host name Indicates the local host internet name. This field is a 255-byte character string that is padded with blanks. The local host internet name is returned to the caller of the SENDTO service. Default: None
ADTRNAME	Remote IP host name Indicates the remote host internet name. this field is a 255-byte character string that is padded with blanks. It is only resolved through internal DNR calls and returned to the caller of the UDP data transfer service routines (RCVFROM and SENDTO) if the ADTOPTNS flag, ADTFDNR, is specified. This is to prevent the DNR call overhead on every UDP data transfer call. Default: None
ADTUCNTX	One word of user context Specifies one arbitrary word of user context to be associated with the endpoint. The information provided is not interpreted by CPT, and is merely saved with other endpoint information. Default: 0 (No user context)
ADTOPTNS	Specifies data transfer options These are the ADT options that apply to UDP data transfer requests: ◆ ADTFDNR – Execute internal DNR calls during UDP data transfer service routine calls (RCVFROM and SENDTO) to resolve remote IP addresses into IP names in the ADTRNAME field. ◆ ADTNBLKR – Do not block on a call to the RCVFROM service (not used by the SENDTO service). Default: None **Note:** These options can be toggled on every UDP data transfer call even if the caller is using the same token.

200801-024040-200100

Network Considerations

The ADT is a common data structure used for both client and server UDP applications. There are common and unique values specified for a particular service request.

This table describes network considerations for Assembler API:

NAME	SERVER CONDITIONS FOR RCVFROM	CLIENT CONTITIONS FOR SENDTO
ADTLPORT	Local server well-known port selected by user application.	Local assigned transport provider port returned to user application.
ADTRPORT	Remote client transport provider port returned to user by user application.	Remote server transport provider well-known port selected by user application.
ADTSRVCE	Local server trans port provider service name selected by user application.	Remote server transport provider service name selected by user application.

NAME	SERVER CONDITIONS FOR RCVFROM	CLIENT CONTITIONS FOR SENDTO
ADTRADDR	Remote IP host address returned to user application.	Remote IP host address selected by or returned to user application. The client must specify this field or ADTRNAME.
ADTLNAME	Local IP host name returned to user application.	Local IP host name returned to user application.
ADTRNAME	Remote IP host name returned to user application only if ADTFDNR is specified in ADTOPCD1.	Remote IP host name selected by or returned to the user application. The client must specify this field or ADTRADDR. If ADTRADDR is used, ADTRNAME will only be returned if ADTFDNR is specified in ADTOPCD1.

Return Codes

The SENDTO service returns a return code in registers 15 and 0 that indicate the results of the execution. These values are in the ADTRTNCD(R15) and ADTDGNCD (R0) within the argument for data transfer. The diagnostic code is optional and indicates the transport provider return code. Read **Appendix C – MESSAGES AND CODES** in the ***CICS Programmer's Toolkit Installation and Administration Guide*** for the return code cross reference table.

This table describes the return codes for the SENDTO service:

RETURN	DIAGNOSTIC CODE	DESCRIPTION
CPTIRCOK		Successful.
CPTEVERS		Control block version number is not supported.
CPTECONN		Requested host/service/port is not found.
CPTEPROT		SENDTO called with a TCP token.
CPTEBUFF		Buffer address and/or length is invalid.
CPTENAPI	Yes	Transport provider API not available.
CPTETERM	Yes	Environment is being terminated.
CPTEPRGE	Yes	CPT interface terminating.
CPTEINTG	Yes	Transport provider integrity error.
CPTEENVR	Yes	Transport provider environment error.
CPTEFRMT	Yes	Transport provider format error.
CPTEPROC	Yes	Transport provider procedure error.
CPTABEND		Abnormal exception occurred.
CPTEOTHR	Yes	An undefined exception occurred.

200801-024040-200100

Examples

This example illustrates a connectionless client. It sends a datagram to a well-known UDP port on a remote host, and then tries to receive it back with a non-blocking RCVFROM call, up to ten times.

```
*-------------------------------------------------------------*
*
*          CICS DSECTS AND WORKING STORAGE SECTION
*
*-------------------------------------------------------------*
CPTPARMS  DS     F             CPT CALLING PARAMETER
CPTADT    T09DADT MF=,PFX=ADT DATA TRANSFER ARGUMENT
CPTIOBUF  DS     XL80          CPT SENDTO/RECVFR BUFFER
LABEL     DFHEIENT CODEREG=(10)
          .
          .
          .

*-------------------------------------------------------------*
*
*          SET UP REMOTE UDP SERVER ADDRESS AND PORT
*
*-------------------------------------------------------------*
          MVC       ADTRPORT,=AL2(7)    UDP ECHO PORT
*
*                                       SET REMOTE ADDRESS TO
*                                       WHITEHOUSE.GOV
*
          MVC       ADTRADDR,=AL1(198,137,240,100)
*-------------------------------------------------------------*
*
*          SENDTO USING UDP DATA TRANSFER, FIRST CREATING A CLIENT
*          TOKEN
*
*-------------------------------------------------------------*
          LA     R04,CPTIOBUF  GET THE ADDRESS OF OUR DATAGRAM
          ST     R04,ADTBUFFA  SET SENDTO BUFFER ADDRESS
          LA     R05,L'CPTIOBUF GET LENGTH OF OUR DATAGRAM
          ST     R05,ADTBUFFL  SET BUFFER LENGTH TO SENDTO
          LA     R03,CPTADT    LOAD DATA TRANSFER ARGUMENT ADDR
          ST     R03,CPTPARMS  SAVE ARGUMENT ADDRESS
*
          T09MCALL SENDTO,PARM=CPTPARMS T09CSNTO ENTRY POINT
*
          LTR    R15,R15       GOOD RETURN CODE?
          BZ     RECVIT        YES, GO RECEIVE IT BACK
          B      EVALERR       NO, GO DIAGNOSE ERROR
RECVIT    DS     0H
*-------------------------------------------------------------*
*
*          RCVFROM USING THE SAME UDP ENDPOINT TOKEN (USING SAME
*          ADT)
*
*-------------------------------------------------------------*
          OI     ADTOPCD1,ADTNBLKR DO NOT BLOCK ON RCVFROM CALL
```

```
                LA      R05,L'CPTIOBUF   GET MAX LENGTH OF DATAGRAM TO
*                                        RCVFROM
                ST      R05,ADTBUFFL     SAVE RCVFROM BUFFER LENGTH
                LA      R06,10           TRY 10 TIMES TO GET IT BACK
RECVLP          DS      0H
*
                T09MCALL RCVFROM,PARM=CPTPARMS T09CRCFR ENTRY POINT
*
                LTR     R15,R15          GOOD RETURN CODE?
                BZ      HOSTALIV            YES, FINISHED REMOTE HOST
*                                           IS THERE
                C       R15,=A(CPTWBLCK) NOTHING AVAILABLE?
                BNE     EVALERR             NO, SOME OTHER ERROR
                BCT     R06,RECVLP          YES, KEEP TRYING
HOSTGONE        DS      0H
```

200801-024040-200100

TAKE

The TAKE service establishes ownership of a connection and associated internal CPT resources. The TAKE service is optional and does not affect an active connection nor does it issue any transport provider requests. This service affects CPT TRUE management routines scheduled on the user's behalf during task termination.

To invoke the TAKE service, a user application is required to first build an AFM and then to issue a call to the TAKE routine. The only valid and required arguments are the AFM version number and the connection token. On completion, a return code is set to indicate success or failure of the request.

This table describes the TAKE service arguments•:

MACRO ID	DSECT NAME	SIZE	CREATED BY
T09DAFM	AFM	34 (X'22')	User application or the LISTEN service

Assembler Data Area

This is what the DSECT control block looks like in Assembler language:

```
Name      Operation    Operands   Description

AFM       DSECT        ,
AFMVERS   DS           H          Version number
AFMFUNC   DS           H          Function code
AFMTOKEN  DS           A          Token (CEP)
          DS           A          Reserved
          DS           F          Reserved
AFMRTNCD  DS           F          Return code
AFMDGNCD  DS           F          Diagnostic code
AFMOPTNS  DS           0F         Facility management option codes
AFMOPCD4  DS           X          Option code 4
AFMOPCD3  DS           X          Option code 3
AFMOPCD2  DS           X          Option code 2
AFMOPCD1  DS           X          Option code 1
          DS           H          Reserved
```

PARAMETER	DESCRIPTION
AFMVERS	Version number Indicates the CPT version number of the argument list used by the calling program. This required field must be set to a binary 2 for this release of CPT. Default: None
AFMFUNC	Function code Indicates the function or callable service ID requested by the application program. This field should not be set by the application, but rather is initialized by the TRUE interface stub program. Default: None

PARAMETER	DESCRIPTION
AFMTOKEN	Connection or endpoint token Specifies a token that represents a TCP connection, a TCP listening endpoint, or a UDP endpoint. A token is created by the TCP connection initiation routines or by the UDP data transfer and endpoint creation routines. Default: 0
AFMRTNCD	Return code Indicates the return code set by the TAKE service. This value is also returned in register 15 and indicates the success or failure of the service. Default: 0
AFMDGNCD	Diagnostic code Indicates the diagnostic code received by the TAKE service for a transport provider request and is not set by the TAKE service. The TAKE service does not issue transport provider requests; hence, it never sets the diagnostic code. Default: 0
AFMOPTNS	Options and events Specifies TAKE processing control options or events detected by the TAKE service. Currently, this facility is reserved for internal processing Default: 0

Completion Information

The TAKE service completes normally when all resources associated with this connection are processed.

On normal return to the application program, the general return code in register 15 (AFMRTNCD) is set to zero (CPTIRCOK). The diagnostic code in register 0 (AFMDGNCD) is always zero.

If the TAKE service completes abnormally, then some resources associated with this connection cannot be successfully transferred from one task to another. The general return code (AFMRTNCD) in register 15 and the diagnostic code (AFMDGNCD) in register 0 indicate the nature of the failure. The diagnostic code (AFMDGNCD) is not used by the TAKE service and no information is returned.

200801-024040-200100

Return Codes

The TAKE service returns a code in register 15 that indicates the results of the execution. This value is in the AFMRTNCD (R15) within the AFM. Read **Appendix C – MESSAGES AND CODES** in the ***CICS Programmer's Toolkit Installation and Administration Guide*** for the return code cross reference table.

This table describes the return codes for the TAKE service:

RETURN	DIAGNOSTIC CODE	DESCRIPTION
CPTIRCOK		Successful.
CPTEVERS		Control block version number is not supported.
CPTETOKN		Specified token is not valid.
CPTENAPI		Transport provider API is not available.
CPTABEND		Abnormal exception occurred.
CPTEOTHR		An undefined exception occurred.

Usage Information

The TAKE service acquires ownership of a connection from one task to another. This service is non-blocking and does not effect any pending transport provider data transfer requests. The association established by the TAKE service lets the CPT properly manage resources during task termination. This ability to GIVE and TAKE ownership of connections offers a range of programming options, while still providing CPT with resource management capabilities.

The TAKE service requires the application to set the AFM version number and token fields. No other AFM fields are referenced.

When a connection is established there are internal CPT resources associated with that connection. CPT is responsible for proper clean-up of those resources on task or transaction termination. These resources include storage allocated by CPT, the API, and the transport provider storage.

A server application is a good example of how the TAKE service benefits a user application. A listening task issues the GIVE service and starts a new transaction to handle data transfer. The data transfer transaction then takes the connection. This sequence prevents a connection from being closed (implicitly by the CPT task termination exit) if the server application terminates. However, if the data transfer transaction terminates without issuing an explicit close (CPT CLOSE service) an implicit close is be scheduled and resource management is handled by the CPT task termination exit.

Additionally, an implicit TAKE facility is implemented with the SEND, RECEIVE, and TRANSLATE services. Any task that issues a SEND, RECEIVE, or TRANSLATE service obtains control of the connection and associated resources. The advantage is that the TAKE service is optional. Even though TAKE can be implicit and therefore optional, Interlink recommends that you issue TAKE to

avoid having a GIVE connection not associated with any transactions. Ownership of a connection and resources provide for clean-up processing during abnormal termination.

The version number (AFMVERS) indicates the CPT release level in which this user application program is written. This required field must be set to a binary 2 and is validated by the TAKE service before processing the request.

The function code (AFMFUNC) indicates the CPT callable service ID. The field is not initialized by a user application program and has little value to the application except for dump analysis. The function code identifies and maps an argument list with the error or trace log and dump analysis.

The token (AFMTOKEN) indicates the connection and internal resources to be processed by the TAKE service. This is a required field and is validated by the TAKE service before processing the request.

The AFMOPTCD field specifies TAKE service processing control options, and provides a mechanism for event notification on return to the application program. Currently, this field is not used by application programs.

Example:

In this example a data processing transaction retrieves the ACM, issues the TAKE service, then performs some required operation. The token is loaded from the ACM and is used by the TAKE service. Register 15 is checked (on return from the TAKE service) and, if successful, processing continues.

```
*
*      Dsect's
*
           T09DACM MF=DSECT     Argument for Connection Management
           T09DAFM MF=DSECT     Argument for Facility Management
*
*          Working storage
*
DFHEISTG   DSECT
           .
           .
           .
ACMARG     DS     XL(ACMLEN)    Argument for Connection Management
TAKEARG    DS     XL(AFMLEN)    Argument for Facility Management
*
*      Entry
*
label      DFHEIENT
           .
           . CICS RETRIEVE command for ACM
           .
           LA     R6,ACMARG     Load ACM address
           USING  T09DACM,R6
           L      R9,ACMTOKEN   Load ACM Token
           DROP   R6            ACM
           .
           .
           .
```

```
*
*         Facility Management TAKE Service
*
TAKE      DS     0H
          LA     R7,TAKEARG    Load TAKE Facility Management arg.
          USING T09DAFM,R7
          MVC    AFMVERS,=H'2'     Set Version number
          ST     R9,AFMTOKEN       Save connection token
          LR     1,R7              Load AFM address in Reg 1
          L      R15,=V(T09FTAKE) Load TAKE service Stub address
          BALR   14,R15            Issue TAKE service
          LTR    15,R15            Test Return Code
          BNZ    ERROR             Non-zero, Process error
          DROP   R7                AFM
          .
          . Application and CPT data transfer (SEND/RECV) processing
          .
ERROR     DS     0H
          .
          . Process and log Application or CPT errors
          .
          .
          . CPT Connection Release
CLOSE     DS     0H
*
*         Terminate Transaction
*
END       DS     0H
          EXECCICS RETURN
```

Example:

In this example a data processing transaction retrieves a CPT token, issues the TAKE service, then performs some required operation. This example uses the T09MCALL macro instruction to set the version number and issue the TAKE service call. Register 15 is checked (on return from the TAKE service) and, if successful, processing continues.

```
*
*         Dsect's
*
          T09DAFM MF=DSECT    Argument for Facility Management
*
*         Working storage
*
DFHEISTG  DSECT
          .
          .
          .
TAKEARG   DS        XL(AFMLEN)Argument for Facility Management
DTTOKEN   DS        A         CPT Data Transfer Token
*
*         Entry
```

SC24040-200100

```
*
label     DFHEIENT
          .
          . CICS RETRIEVE command for CPT Token
          .
          L         R5,DTTOKEN    Load DT Token

          .
          .
          .
*
*         Facility Management TAKE Service
*
TAKE      DS        0H
          LA        R8,TAKEARG    Load TAKE argument address
          USING     T09DAFM,R8
          ST        R5,AFMTOKEN       Save connection token
          T09MCALL  TAKE,PARM=TAKEARGIssue Take Service request
          LTR       15,R15            Test Return Code
          BNZ       ERROR             Non-zero, Process error
          DROP      R8                AFM

          .
          . Application and CPT data transfer (SEND/RECV) processing
          .
ERROR     DS     0H
          .
          . Process and log Application or CPT errors
          .
          .
          . CPT Connection Release

CLOSE     DS        0H
*
*         Terminate Transaction
*
END       DS        0H
          EXEC      CICS RETURN
```

200801-024040-200100

Example:

In this example a data processing transaction retrieves an ACM, and performs some required operation. The token is loaded from the ACM and used by the CPT SEND and/or RECEIVE services. The difference between this example and the first example is that no TAKE service is explicitly issued by the application. The SEND and/or RECEIVE services implement the TAKE service internally, so an explicit TAKE service request can be omitted.

```
*
*         Dsect's
*
          T09DAFM MF=DSECT     Argument for Facility Management
          T09DACM MF=DSECT     Argument for Connection Management
*
*         Working storage
*
DFHEISTG  DSECT
          .
          .
          .
TAKEARG   DS      XL(AFMLEN)   Argument for Facility Management
ACMARG    DS      XL(ACMLEN)   Argument for Connection Management
*
*         Entry
*
label     DFHEIENT
          .
          . CICS RETRIEVE command for ACM
          .
          LA      R8,ACMARG          Load ACM address
          USING   T09DACM,R8
          L       R10,ACMTOKEN       Load ACM Token
          DROP    R8                 ACM
          .
          . Application and CPT data transfer (SEND/RECV) processing
          .
ERROR     DS      0H
          .
          . Process and log Application or CPT errors
          .
          .
          . CPT Connection Release
CLOSE     DS      0H
*
*         Terminate Transaction
*
END       DS      0H
          EXEC    CICS RETURN
```

SC34-5455-02 024040-200100

TRANSLATE

The TRANSLATE service translates data between EBCDIC and ASCII character sets. CPT is customized with a default translation table; however, applications can override the default. The TRANSLATE service does not affect an active connection nor issue any transport provider requests.

To invoke the TRANSLATE service, a user application is required to first build an Argument for Data Translation (AXL) and then to issue a call to the TRANSLATE routine. The AXL is required to contain the version number, connection token, user buffer address and length, and type or direction of translation requested. Additional arguments for application specific translation tables are supported. When the TRANSLATE service completes, the buffer contents are converted into the corresponding characters and a return code is generated indicating the status of the request.

This table describes the TRANSLATE service arguments:

MACRO ID	DSECT NAME	SIZE	CREATED BY
T09DAXL	AXL	32 (X'20')	User application

Assembler Data Area

This is what the DSECT control block looks like in Assembler language:

```
Name      Operation    Operands  Description

AXL       DSECT        ,
AXLVERS   DS           H         Version number
AXLFUNC   DS           H         Function code
AXLTOKEN  DS           A         Token (CEP)
AXLSADDR  DS           A         Source text address
AXLSLENG  DS           F         Source text length
AXLRTNCD  DS           F         Return code
AXLDGNCD  DS           F         Diagnostic code
AXLXMODE  DS           0H        Character set mode
AXLXMOD2  DS           X
AXLXMOD1  DS           X
AXLXSBCS  EQU          X'00'     * Single byte character set
AXLXDBCS  EQU          X'01'     * Double byte character set
AXLMXMIXD EQU          X'02'     * Mixed SBCS/DBCS character set
AXLMNUMS  EQU          X'04'     * Numeric character set
AXLLTYPE  DS           0H        Translation type request
AXLLTYP2  DS           X
AXLLTYP1  DS           X
AXLTATOE  EQU          X'01'     * Translate ASCII to EBCDIC
AXLTETOA  EQU          X'02'     * Translate EBCDIC to ASCII
AXLTAUPC  EQU          X'04'     * Translate ASCII to uppercase
AXLTEUPC  EQU          X'08'     * Translate EBCDIC to uppercase
AXLTABLE  DS           A         Address of user translate table
```

200801-024040-200100

PARAMETER	DESCRIPTION
AXLVERS	Version Indicates the CPT version number of the argument used by the calling program. This required field must be set to a binary 2 for this release of CPT. Default: None
AXLFUNC	Function code Indicates the function or callable service ID requested by the application program. This field should not be set by the application, but is initialized by the TRUE interface stub. Default: None
AXLTOKEN	Connection token Specifies a token that represents a TCP connection or UDP endpoint. Default: 0
AXLSADDR	Source text buffer address Indicates the address of the user data buffer to be translated. This required field is a contiguous segment of storage accessible to the user task. The storage area can be aligned on any boundary convenient to the application program. Default: 0
AXLSLENG	Source text buffer length Indicates the length (in bytes) of user data buffer in the storage area, as identified by the AXLSADDR field. This is a required field. A zero value causes the request to fail. Default: 0
AXLRTNCD	Return code Indicates the return code set by the TRANSLATE service. This value is also returned in register 15 and indicates the success or failure of the service. Default: 0
AXLDGNCD	Diagnostic code Indicates the diagnostic code set by the service request. This value specifies a unique number associated with the return code and identifies the translation error. Default: 0
AXLXMODE	Specifies TRANSLATE service translation mode or character set. ◆ AXLXSBCS – Indicates single-byte character set translation. ◆ AXLXDBCS – Indicates double-byte character set translation. This option is currently not supported. ◆ AXLXMIXD – Indicates mixed mode character set translation. This mode specifies single- and double-byte translation. This option is currently not supported. ◆ AXLXNUMS – Indicates numeric set translation. This option is currently not supported. Default: AXLXSBCS

PARAMETER	DESCRIPTION
AXLLTYPE	Specifies TRANSLATE service translation type or direction. ◆ AXLTATOE – Indicates ASCII to EBCDIC translation. ◆ AXLTETOA – Indicates EBCDIC to ASCII translation. ◆ AXLTAUPC – Indicates ASCII to uppercase ASCII translation. ◆ AXLTEUPC – Indicates EBCDIC to uppercase EBCDIC translation. Default: None
AXLTABLE	Address of user translation table. Default: None

Completion Information

The TRANSLATE service completes normally when the data is translated into the corresponding character set representation.

On normal return to the application program, the general return code (AXLRTNCD) in register 15 is set to zero (CPTIRCOK). The diagnostic code (AXLDGNCD) in general register 0 is set to zero.

If the TRANSLATE service completes abnormally, an error associated with translation occurred. The general return code (AXLRTNCD) in register 15 and the diagnostic code (AXLDGNCD) in register 0 indicate the nature of the failure.

200801-024040-200100

Return Codes

The TRANSLATE service returns a code in registers 15 and 0 that indicates the results of the execution. These values are in the AXLRTNCD (R15) and AXLDGNCD within the AXL. Read **Appendix C – MESSAGES AND CODES** in the ***CICS Programmer's Toolkit Installation and Administration Guide*** for the return code cross reference table.

This table describes the TRANSLATE service return codes:

RETURN	DIAGNOSTIC CODE	DESCRIPTION
CPTIRCOK		Successful.
CPTEVERS		Control block version number is not supported.
CPTETOKN		Specified token is not valid.
CPTEBUFF		Buffer address and/or length is invalid.
CPTECHAR		Translation table character set is invalid.
CPTEMODE		Translate mode specification is invalid.
CPTEFRMT		Format or specification error.
CPTENAPI		Transport provider API is not available.
CPTABEND		Abnormal exception occurred.
CPTEOTHR		An undefined exception occurred.

Usage Information

The TRANSLATE service translates data between EBCDIC and ASCII. The requirement for translation is application dependent.

The version number (AXLVERS) indicates the CPT release level in which this user application program is written. This required field must be set to a binary 2 and is validated by the TRANSLATE service before processing the request.

The function code (AXLFUNC) indicates the CPT callable service ID. The field is not initialized by a user application program and has little value to the application except for dump analysis. The function code can identify and map an argument list with the error or trace log and dump analysis.

The token (AXLTOKEN) indicates the connection associated with this translation request. This field is required; however, no transport provider requests are issued. The token is used for internal logging support requirements.This required field is validated by the TRANSLATE service before processing the request.

The AXLXMODE field specifies the character set mode. This field sets single, double or mixed character set translation. Currently, only single-byte character set translation, which is the default, is supported.

20-024040-200100

The AXLLTYPE field specifies the translation direction. This required field indicates EBCDIC to ASCII, or ASCII to EBCDIC. Additionally, characters can be transacted into the corresponding uppercase values.

Example:

In this example an established connection sends data to an ASCII host. The data is translated by the application and then sent over the connection. The version number, token, data buffer address and length, and translation type are set in the AXL. The default translation mode of Single-Byte Character Set (SBCS) is selected. Register 15 is checked to determine the success or failure to the request.

```
*
*         Dsect's
*
          T09DAXL MF=DSECT    Argument for Data Translation
          T09DADT MF=DSECT    Argument for Data Transfer
*
*      Working storage
*
DFHEISTGDSECT
          .
          .
          .
DATA      DS     CL80             Data Buffer
XLATARG   DS     XL(AXLLEN)       Argument for Data Translation
SENDARG   DS     XL(ADTLEN)       Argument for Data Transfer
*
*         Entry
*
label     DFHEIENT
          .
          . CPT Connection or Facility Management Service request
          .
          L      R9,token         Load Connection Token
GETDATA   DS     0H
          .
          . Application to place data in buffer (DATA)
          .
          LA     R7,XLATARG       Load TRANSLATE argument addr
          USING  T09DAXL,R7
          MVC    AXLVERS,=H'2'    Set Version number
          ST     R9,AXLTOKEN      Save connection token
          LA     R2,DATA          Load address of data buffer
          ST     R2,AXLSADDR      Save Source Text Buffer address
          LA     R3,L'DATA        Save connection token
          ST     R3,AXLSLENG      Save Source Text Buffer length
          MVI    AXLLTYP1,AXLTATOE ASCII to EBCDIC translation
          LR     R1,R7            Load AXL address in Reg 1
          L      R15,=V(T09FXLAT) Load TRANSLATE stub routine address
          BALR   R14,R15          Issue TRANSLATE service
          LTR    R15,R15          Test Return Code
          BNZ    ERROR            Non-zero, process error
          DROP   R7               AXL
```

```
          .
          . CPT SEND service request
          .
          B      GETDATA          Process more data
ERROR     DS     0H
          .
          . Process and log Application or CPT errors
          .
          .
          . CPT Connection Release

CLOSE     DS     0H
*
*         Terminate Transaction
*
END       DS     0H
          EXEC   CICS RETURN
```

Example:

In this example an established connection is receiving data from a ASCII host. The application receives data from the network connection and then is required to translate the data into EBCDIC. The token, data buffer address and length, and translation type are set in the AXL. The `T09MCALL` macro instruction sets the version number and issues the `TRANSLATE` service call. Register 15 is checked to determine the success or failure of the request.

```
*
*         Dsect's
*
          T09DAXL MF=DSECT        Argument for Data Translation
          T09DADT MF=DSECT        Argument for Data Transfer
*
*         Working storage
*
DFHEISTG  DSEC
          .
          .
          .
DATA      DS     CL80             Data Buffer
XLATARG   DS     XL(AXLLEN)       Argument for Data Translation
RECVARG   DS     XL(ADTLEN)       Argument for Data Transfer
*
*         Entry
*
label     DFHEIENT
          .
          . CPT Connection or Facility Management Service request
          .
          L      R9,AXLTOKEN      Load Connection Token
GETDATA   DS        0H
          .
          .
          .
```

SC24-5040-200100

```
                LA      R10,ADTARG        Load Data Transfer argument addr
                USING   T09DADT,R10

                .
                . CPT RECEIVE service request
                .

                LA      R7,XLATARG        Load TRANSLATE argument addr
                USING T09DAXL,R7
                ST      R9,AXLTOKEN       Save connection token
                L       R2,ADTBUFFA       Load Receive data address
                ST      R2,AXLSADDR       Save Source Text Buffer address
                L       R3,ADTBUFFL       Load Receive data length
                ST      R3,AXLSLENG       Save Source Text Buffer length
                MVI     AXLLTYP1,AXLTETOA   EBCDIC to ASCII translation
                T09MCALL XLAT,PARM=XLATARG Issue TRANSLATE service
*                                           request
                BALR    R14,R15         Issue TRANSLATE service
                LTR     R15,R15         Test Return Code
                BNZ     ERROR           Non-zero, process error
                DROP    R7              AXL

                .
                . Application processes data
                .

                B       GETDATA         Process more data

ERROR           DS      0H
                .
                . Process and log Application or CPT errors
                .

                .
                . CPT Connection Release

                CLOSE   DS  0H

*
*               Terminate Transaction
*
END             DS      0H
                EXEC    CICS RETURN
```

200801-024040-200100

T09MCALL

The T09MCALL macro instruction calls the CICS/API service. This macro sets the version number, loads the address of the argument in register 1, and executes the service.

Use this syntax to call the T09MCALL service:

```
T09MCALL service
    [,PARM=service argument]
    [,VERSION=argument version number]
```

PARAMETER	DESCRIPTION
service	T09MCALL can call any of the CPT services: ◆ CLOSE – Keyword that specifies the API CLOSE service. ◆ CONNECT – Keyword that specifies the API CONNECT service (TCP only). ◆ GIVE – Keyword that specifies the API GIVE service. ◆ LISTEN – Keyword that specifies the API LISTEN service (TCP only). ◆ RCVFROM – Keyword that specifies the API RCVFROM service (UDP only). ◆ RECEIVE – Keyword that specifies the API RECEIVE service (TCP only). ◆ SEND – Keyword that specifies the API SEND service (TCP only). ◆ SENDTO – Keyword that specifies the API SENDTO service (UDP only). ◆ TAKE – Keyword that specifies the API TAKE service. ◆ TRANS – Keyword that specifies the API TRANSLATE service.
PARM	*service argument* Indicates the address of a storage area containing the service request argument. The address of the argument is loaded into register 1. This field is optional and the argument is assumed to be in register 1.
VERSION	*argument version number* Specifies the CPT version number for this argument. Currently, binary 2 is the only valid version number supported. This field is optional and the argument version number is set to 2.

20-024040-200100

Return Codes

The T09DRTCD macro instruction resolves API service return codes.

```
          MACRO
          T09DRTCD

Name      Operation Operands  Description

                    SUCCESSFUL COMPLETION SETTING

CPTIRCOK  EQU       X'00'     Request completed successfully.

                    WARNING LEVEL MESSAGES

CPTWTIMO  EQU       X'01'     Timed receive service call timed
                              out
CPTWNEGO  EQU       X'04'     System limits applied to buffer and
                              queue sizes
CPTWBLCK  EQU       X'06'     Receive would block (No data currently
                              avail.)
CPTWNEOM  EQU       X'08'     UDP - This is not the whole
                              datagram
CPTWNSEP  EQU       X'A'      SEP type receive found no separator
                              characters
CPTWEXCP  EQU       X'0F'     TPL exceptional condition

                    CONTROL BLOCK ARGUMENT ERRORS

CPTEVERS  EQU       X'11'     Control block version number not
                              supported
CPTECONN  EQU       X'12'     Req host/service/port connection
                              not found
CPTEPROT  EQU       X'13'     Specified protocol not supported
CPTETOKN  EQU       X'14'     Specified data transfer token is
                              invalid
CPTEBUFF  EQU       X'15'     Buffer address and/or length
                              invalid
CPTECHAR  EQU       X'16'     Translate character set is invalid
CPTEMODE  EQU       X'17'     Translate mode specification is
                              invalid
CPTECOPT  EQU       X'18'     CLOSE mode specification is
                              invalid
CPTETABL  EQU       X'19'     Specified translate table not
                              correct
CPTETRID  EQU       X'1A'     Designated transaction ID cannot be
                              started
CPTETIME  EQU       X'1B'     RECV time out value must be > zero
CPTESEP#  EQU       X'1C'     RECV type SEP requires # SEP
                              characters = 1 or 2
CPTEOPTN  EQU       X'1D'     RECV options combination is
                              invalid
CPTEOPRL  EQU       X'1E'     RECV option not supported by transport
                              carrier
CPTEFRMT  EQU       X'1F'     Other TPL format or specification
                              error

                    LOCAL ENVIRONMENT ERRORS

CPTEPBSY  EQU       X'21'     Selected port is busy with active
                              server
```

200801-024040-200100

```
CPTENAPI  EQU      X'22'     SNS/API/CICS not fully available,
                             retry
CPTENAVL  EQU      X'23'     Requested facility is not
                             avaialable
CPTEDRAN  EQU      X'24'     Environment is being drained
CPTETERM  EQU      X'28'     Environment is being terminated
CPTEENVR  EQU      X'2F'     Other TPL environmental condition

                        CONNECTION ERRORS

CPTERLSE  EQU      X'41'     Orderly release of remote connection
                             requested
CPTEDISC  EQU      X'44'     Rconnection not available or
                             aborted
CPTEPRGE  EQU      X'48'     Remote connection environment
                             terminating
CPTEINTG  EQU      X'4F'     Other TPL connection/data
                             integrity error

              ANY TPL SEQUENCE OR PROCEDURAL ERROR

CPTEPROC  EQU      X'8F'

       ABNORMAL ENVIRONMENTAL ERROR FORCING CI00 ABEND

CPTABEND  EQU      X'FE'

          ANY OTHER CURRENTLY UNDEFINED CONDITION*

CPTEOTHR  EQU      X'FF'
```

CODE NAME	DESCRIPTION
CPTIRCOK	Request completed successfully Indicates the requested service completed successfully.
CPTWTIMO	Timed RECEIVE service call timed out See the following RECEIVE service options: ADTTYPLL ADTTYPSP ADTTMRCV ASTTMPRT
CPTWNEGO	System limits applied to buffer and queue sizes A connection management service returned modified API buffering values. The ACMQSEND, ACMMSEND, ACMQRECV or ACMMRECV values were modified during transport provider connection negotiation. This return code does not generally indicate a serious error, but rather a warning.
CPTWBLCK	RECEIVE would block (no data currently available) No data was available on a TCP connection if the call was to the RECEIVE service, or no datagrams were available at a UDP endpoint if the call was to the RCVFROM service. In both cases, ADTOPCD1 was set to ADTNBLKR.
CPTWNEOM	This is not the whole datagram On a RCVFROM service call, ADTBUFFA for a length of the ADTBUFFL was not large enough to hold the entire datagram. Subsequent RCVFROM calls will be required to retrieve the rest of the datagram.

CODE NAME	DESCRIPTION
CPTWNSEP	On a RECEIVE service call with the SEP option specified (ADTTYPSP) no indicated separator sequence was found in the data. Check user data and make sure the ADTTIMEO value was long enough to receive all of the data.
CPTWEXCP	Transport provider exceptional condition A transport provider exceptional error has occurred. Review diagnostic code for additional information.
CPTEVERS	Argument version number not supported Indicates the version number specified is not supported. The service request is not executed.
CPTECONN	Requested host/service port connection not found A connection management service request failed due to an invalid or unresolved host, service name or port number specification. The service request is not executed.
CPTEPROT	Specified protocol not supported A connection management service request failed due to an invalid or unsupported protocol specification. The protocol specified was not TCP or UDP. The service request is not executed.
CPTETOKN	Specified token is invalid Specified token in service argument is invalid. The service request is not executed.
CPTEBUFF	Buffer address and/or length invalid Specified buffer address and/or length is invalid. A data transfer or data translation service request failed verification of buffer address or length. The service request is not executed.
CPTECHAR	Translate character set is invalid Specified translation character set is invalid. The translation service request is not executed.
CPTEMODE	Translate mode specification is invalid Specified translation mode is invalid. The Translation service request is not executed.
CPTECOPT	CLOSE mode specification is invalid An invalid or unknown CLOSE option (ACLOPCD1) was specified on a call to the CLOSE service.
CPTETABL	Specified translate table not correct An invalid character set table was specified on a call to the TRANSLATE service.
CPTETRID	Designated transaction ID cannot be started The LISTEN service attempted to start a transaction that failed. The LISTEN service request is terminated and the well-known server port is released.
CPTETIME	RECEIVE service timeout value must be greater than zero Specifying any of these options on a RECEIVE call with an ADTTIMEO=0 will result in CPTETIME being returned in ADTRTNCD: ADTTYPLL ADTTYPSP ADTTMRCV ADTTMPRT

200801-024040-200100

CODE NAME	DESCRIPTION
CPTESEP#	RECEIVE type SEP requires # SEP chars = 1 or 2 On a RECEIVE service call with the SEP option specified (ADTTYPSP), ADTSEP# was not equal to 1 or 2.
CPTEOPTN	RECEIVE options combination is invalid On a RECEIVE service call, more than one of these options was specified: ADTNBLKR ADTTYPLL ADTTYPSP ADTTMRCV ADTTMPRT
CPTEOPRL	RECEIVE option not supported by transport carrier You have attempted a CPT function that is not supported by the release of SNS/TCPaccess that you are running.
CPTEFRMT	Other transport provider format or specification error A transport provider format or specification error occurred. Review diagnostic code for additional information. The service request failed to execute successfully.
CPTEPBSY	Selected port is busy with active server The LISTEN service attempted to establish a connection to the local transport provider but the specified port was already allocated. The LISTEN service request failed to execute.
CPTENAPI	API, transport provider or CICS not available The transport provider, API communication subsystem or CICS is not available. Review diagnostic code for additional information. The service request failed to execute.
CPTEDRAN	Transport provider or API is being drained The transport provider or API communication subsystem is being drained. This is an indication that the CPT programming interface may be terminating.
CPTETERM	Transport provider or API is being terminated The transport provider or API communication subsystem is being terminated. This is an indication that the CPT programming interface is terminated. All connections have been closed and no communication is available.
CPTEENVR	Other transport provider environment error A transport provider environment error occurred. Review diagnostic code for additional information. The service request failed to execute successfully.
CPTERLSE	Orderly release of full duplex connection indicated Indicates that the remote host has released its half of the full duplex connection.The connection is still available for communication, but the user application will not receive data for the remote host.
CPTEDISC	Remote connection not available or aborted Indicates that a connection request failed or an established connection was aborted. The connection request is not established or an established connection is terminated.
CPTEPRGE	CICS Programmer's Toolkit interface terminated The CPT interface was terminated. The CPT termination transaction was initiated by either the termination transaction or CICS shutdown PLT entry.

20 024040-200100

CODE NAME	DESCRIPTION
CPTEINTG	Other transport provider integrity error A transport provider integrity error occurred. Review diagnostic code for additional information. The service request failed to execute successfully.
CPTEPROC	Other transport provider procedure error A transport provider procedure error occurred. Review diagnostic code for additional information. The service request failed to execute successfully.
CPTABEND	Abnormal environmental error The service request experienced an abnormal termination condition. The service request failed to execute. The CICS dump data set contains additional information related to the abend.
CPTEOTHR	Any other currently undefined condition An unknown condition was detected. Review the diagnostic code for additional information. The service request failed to execute.

200801-024040-200100

CICS Storage Requirements

All CICS storage is shared and allocated from above the 16M line, if possible. Storage requirements for a TCP connection or a UDP endpoint within the CICS address space are:

- `1080 + ((72 + xxxMSEND) * xxxQSEND) + ((72 + xxxMRECV) * xxxQRECV)`

  where `xxx` is either ACM for TCP connections or ADT for UDP endpoints.

Note: For TCP server applications, an additional 1080-byte overhead is required to allocate the listening token/socket. This computation accounts only for the TCP connection that is subsequently established by a listening token/socket or by the storage for a UDP endpoint.

Note: The send buffer queue is allocated only if an application calls the `SEND` or `SENDTO` services. Determining the `xxxMSEND` and `xxxQSEND` values is explained below. The receive buffer queue is allocated only if an application calls the `RECEIVE` or `RCVFRM` services. Determining the `xxxMRECV` and `xxxQRECV` values is explained below.

Note: The receive buffer queue is allocated only if an application calls the `RECEIVE` or `RECVFROM` services. Determining the `xxxMRECV` and `xxxQRECV` values is explained below.

The maximum send and receive data sizes, the maximum send and receive buffer sizes, and the send and receive queues' sizes (maximum number of outstanding requests per endpoint) are determined initially by the `ACPCONFG` macro, `ACFTIB`. Read the ***SNS/TCPaccess Customization Guide***, pages A-70 and A-71, for information. The TCP maximum data size default that can be sent or received is 32K. The maximum data buffer size default is 64K. CPT negotiates with SNS/PCaccess to determine these values:

- Determine final `xxxQSEND` value:

```
If xxxSEND is not specified,
       use the lesser of 4 or the value in DFQSEND
   else
       use the lesser of the value specified in xxxQSEND or
       DFQSEND.
```

- Determine final `xxxQRECV` value:

```
If xxxQRECV is not specified,
       use the lesser of 4 or the value in DFGRECV
   else
       use the lesser of the value specified in xxxQRECV or
       DFQRECV.
```

- Determine final `xxxMSEND` value:

```
If xxxMSEND is not specified,
       use the lesser of 4096 or the value in MXLTSND
   else
```

20-024040-200100

```
        use the lesser of the value specified xxxMSEND or
        MXLTSND
    either of which must be less than the following
    computation:
        (the lesser of DFLSEND or 61440) / the final
        xxxQSEND value
    else
        use the quotient from this computation as thefinal
        xxxMSEND value.
```

- **Determine final xxxMRECV value:**

```
If xxxMRECV is not specified,
        use the lesser of 4096 or the value in MXLTRCV
    else
        use the lesser of the value specified in xxxMSEND or
        MXLTRCV
either of which must be less than the following
computation:
(the lesser of DFLRECV or 61440) / the final xxxQRECV value
    else
        use the quotient from this computation as the final
        xxxMRECV value.
```

Chapter 3
C SUBROUTINE CALLS

This chapter describes the C subroutine calls of CICS/API. It contains these callable routines:

- CLOSE
- CONNECT
- GIVE
- LISTEN
- RCVFROM
- RECEIVE
- SEND
- SENDTO
- TAKE
- TRANSLATE
- T09KCALL

It also describes the header file T09KCALL, return codes of the T09KSRCS header file, and the CICS storage requirements for a TCP connection or a UDP endpoint.

All messages are in **Appendix C – MESSAGES AND CODES** in the ***CICS Programmer's Toolkit Installation and Administration Guide***.

200801-024040-200100

CLOSE

The CLOSE service closes an established connection. Both orderly and abortive termination options are supported. The CLOSE service performs all associated functions required for CPT resource clean-up.

To invoke the CLOSE service, a user application is required to first build an Argument for Close (ACL) and then to issue a call to the CLOSE routine. Valid arguments include the ACL version number, connection token, and termination options. On completion, a return code indicates success or failure of the request.

This table describes the arguments for the CLOSE service:

INCLUDE STATEMENT	STRUCT NAME	SIZE	CREATED BY
t09ksacl.h	acl_stru	30 (X'1E')	User application

Structure

This is the structure of the CLOSE service in C language:

```
#ifndef __t09ksacl__
 #define __t09ksacl__
 /* t09ksacl.h --  Declare CPT-dependent structures     */

 /*     ACCCSECT UPDTID=A1000000                        */

 /*                                                     */
 /*  Define Connection Release (ACL) Structure          */
 /*                                                     */

 typedef struct
     {
    short acl_vers     ;  /* acl block version number  */
    short acl_func     ;  /* Request function type     */
    int  *acl_token    ;  /* Data transfer token       */
    void *acl_rsvd1    ;  /* reserved  (in use)        */
    int   acl_rsvd2    ;  /* reserved  (in use)        */
    int   acl_rtncd    ;  /* Return code               */
    int   acl_dgncd    ;  /* Diagnostic code           */
    char  acl_oprsv [3] ;  /* Reserved options          */
    char  acl_optns    ;  /* Termination options       */
    short acl_rsvd3    ;  /* reserved  (in use)        */
     } acl_stru;

     /*-------------------------------------------*/
       /*       CONTROL BLOCK VERSION NUMBER       */
     /*-------------------------------------------*/
 #define ACL_VERSN       2

     /*-------------------------------------------*/
       /*        TERMINATION TYPE REQUEST          */
     /*-------------------------------------------*/
 #define ACLOPT_ORDER 0x00   /*     ORDERLY RELEASE     */
```

200PN1-024040-200100

```
#define ACLOPT_ABORT 0x01   /*      ABORTIVE STOP      */

    /*-------------------------------------------*/
     /*       CPT CONNECTION RELEASE SERVICE      */
    /*-------------------------------------------*/
#if defined (IBMC)
 #pragma linkage(t09fclos,OS)
#else
 #define t09fclos __asm_t09fclo
#endif

 void t09fclos ();

#endif
```

PARAMETER	DESCRIPTION
acl_vers	Version number Indicates the CPT version number of the argument list used by the calling program. This required field must be set to ACL_VERSN for this release of CPT. Default: None
acl_func	Function code Indicates the function or callable service ID requested by the application program. This field is not set by the application, but is initialized by the *Task-Related User Exit* (*TRUE*) interface stub program. Default: None
acl_token	Connection token Specifies a token that represents the connection. A token is obtained from a connection management request. The token is required. Default: None
acl_rtncd	Return code Indicates the return code set by the CLOSE service. This value indicates the success or failure of the service. Default: 0
acl_dgncd	Diagnostic code Indicates the diagnostic code set by the service request. This value generally indicates a transport provider return code. Default: 0

20-024040-200100

PARAMETER	DESCRIPTION
acl_optns	Specifies CLOSE service processing control options. These are the options supported: ◆ ACLOPT_ORDER – Indicates a graceful termination. This option implements orderly release of the TCP/IP connection. This is the preferred option for terminating a connection, and is used when processing has completed successfully. ◆ ACLOPT_ABORT – Indicates abortive termination. This option implements a disconnect or reset of the TCP/IP connection. This option is generally used after an unrecoverable application error has occurred. **Note:** The notion of orderly or abortive CLOSE for a UDP endpoint is meaningless and the options specified when calling CLOSE for a UDP token are not important. CPT knows if the token is UDP and will close it properly. Default: ACLOPT_ORDER

Completion Information

The CLOSE service completes normally when the connection is terminated and associated resources are released. Graceful termination waits for all pending transport provider asynchronous SEND and RECEIVE requests to complete. Graceful termination also waits for both ends of the full-duplex connection to close. Abortive termination closes the transport provider connection without regard to pending transport provider requests. Abortive termination can lead to possible data loss and should be used only when data integrity is not required.

On normal return to the application program, the general return code in acl_rtncd is set to zero (CPTIRCOK). The diagnostic code in acl_dgncd is always zero.

If the CLOSE service completes abnormally, some user data may be lost. The general return code in acl_rtncd, and the diagnostic code in acl_dgncd, indicate the nature of the failure. The diagnostic code (acl_dgncd) may contain a specific code that identifies a particular transport provider error.

200801-024040-200100

Return Codes

The CLOSE service return and diagnostic codes indicate the result of the execution. These values are in the acm_rtncd and acm_dgncd within the connection management argument. The diagnostic code is optional and indicates the transport provider return code. Read **Appendix C – MESSAGES AND CODES** in the ***CICS Programmer's Toolkit Installation and Administration Guide*** for the return code cross reference table.

This table describes the CLOSE service return codes:

RETURN	DIAGNOSTIC CODE	DESCRIPTION
CPTIRCOK		Successful.
CPTEVERS		Control block version number is not supported.
CPTETOKN		Specified token is not valid.

RETURN	DIAGNOSTIC CODE	DESCRIPTION
CPTEPRGE	Yes	CPT Interface is terminating.
CPTENAPI	Yes	Transport provider API is not available.
CPTETERM	Yes	Environment is being terminated.
CPTERLSE	Yes	Release indication.
CPTEDISC	Yes	Disconnect indication.
CPTEINTG	Yes	Transport provider API integrity error.
CPTEENVR	Yes	Transport provider API environment error.
CPTEFRMT	Yes	Transport provider API format error.
CPTEPROC	Yes	Transport provider API procedure error.
CPTABEND		Abnormal exception occurred.
CPTEOTHR	Yes	An undefined exception occurred.

Usage Information

The CLOSE service terminates an established transport provider connection, and releases associated resources. Established transport provider connections can half of a TCP connection, a TCP listening endpoint, or a UDP endpoint, and are represented by a token.

The CLOSE service utilizes the ACL. The CLOSE service requires the application to set the ACL version number and token fields. Optional control information related to termination processing can be specified.

The version number (acl_vers) indicates the CPT release level in which this user application program is written. This required field must be set to ACL_VERSN and is validated by the CLOSE service before it processes the request.

The function code (acl_func) indicates the CPT callable service ID. The field is not initialized by a user application program and has little value to the application except for dump analysis. The function code can identify and map an argument list with the error or trace log and dump analysis.

The token (acl_token) indicates the connection and internal resources that are to be released. This is a required field and is validated by the CLOSE service before processing the request.

The acl_optns field specifies CLOSE processing control options and provides a mechanism for event notification on return to the application program. Currently, ACLOPT_ORDER and ACL_ABORT are the only two options supported; no facility exists for CLOSE event notification, except by way of return code values.

If the option code ACLOPT_ORDER is selected, the CLOSE service completes all pending transport provider requests. These requests represent previous asynchronous SEND and/or RECEIVE services that have neither completed yet, nor had their completion checked. This may require the CLOSE service to block

the application. This option then performs an orderly release of the TCP/IP connection. This is the preferred mechanism for connection termination.

If the option code `ACL_ABORT` is selected, the `CLOSE` service terminates the connection and no attempt is made to preserve data in transit. The remote user receives a disconnect indication.

200R01-024040-200100

Example:

In this example a connection is established, data is processed, and the connection is closed. The token is loaded from the Argument for Connection Management (ACM) and used by all of the following CPT service requests. Graceful termination is requested. The `acl_rtncd` is checked to determine `CLOSE` service request status.

```
#include  <t09ksacl.h>
#include  <t09ksadt.h>
#include  <t09ksacm.h>
#include  <t09ksrcs.h>

void main()
{
    struct acl_stru
        cpt_acl = {2, 0, 0, NULL, NULL, 0, 0, 0, 0};

    /*
    *  CPT Connection Management service request
    */

    cpt_adt.adt_token = cpt_acm.acm_token;

    /*
    *  Data Processing Routine
    */

    while (data)
    {
        /*
        * Application and CPT Data Transfer (SEND/RECEIVE) processing
        */
        }

    /*
    *  Orderly Release connection
    */

    Close:
    cpt_acl.acl_token = cpt_acm.acm_token;
    cpt_acl.acl_optns = ACLOPT_ORDER;
    t09fclos (&cpt_acl);
    if (cpt_acl.acl_rtncd != 0)
    {
        /* process and log error */
    }

    /*
    *  Terminate Transaction
    */

    End:
    EXEC CICS RETURN;
}
```

024040-200100

Example:

In this example an established connection receives an error while processing data and aborts the connection. The ACL version and token are specified. The ACL abort option is selected to indicate the type of connection termination required. The return code is checked to determine CLOSE service request status.

```
#include  <t09ksacl.h>
#include  <t09ksacm.h>
#include  <t09ksadt.h>
#include  <t09ksrcs.h>

void main()
{
    struct acl_stru
        cpt_acl = {2, 0, 0, NULL, NULL, 0, 0, 0, 0};

    /*
    *  CPT Connection Management service request
    */

    cpt_adt.adt_token = cpt_acm.acm_token;

    /*
    *  Data Processing Routine
    */

    while (data)
    {
        /*
        * Application and CPT Data Transfer (SEND/RECEIVE) processing
        */

    if (cpt_adt.adt_rtncd != 0);
        break;
    }

    /*
    *  Abortive Release connection
    */

    Abort:
    cpt_acl.acl_token = cpt_acm.acm_token;
    cpt_acl.acl_optns = ACLOPT_ABORT;
    t09fclos (&cpt_acl);
    if (cpt_acl.acl_rtncd != 0)
    {
        /* process and log error */
    }

    /*
    *  Terminate Transaction
    */

    End:
    EXEC CICS RETURN;
}
```

200801-024040-200100

Example:

This example terminates a single-thread server application. A server application can contain two CPT connections; the first is for the data transfer connection and the second for the server or listening connection. The tokens are determined for the ACM. A CLOSE service is issued for both of the connections. The return code is checked to determine CLOSE service request status.

```
#include  <t09ksacl.h>
#include  <t09ksadt.h>
#include  <t09ksrcs.h>
#include  <t09ksacm.h>

void main()

{
    struct acl_stru
        cpt_acl = {2, 0, 0, NULL, NULL, 0, 0, 0, 0};

    /*
    *  CPT Connection Management service request
    */

    cpt_adt.adt_token = cpt_acm.acm_token;

    /*
    *  Data Processing Routine
    */

    while (data)
    {
    /*
        * Application and CPT Data Transfer (SEND/RECEIVE) processing
    */
    }

    /*
    *  Orderly Release Data Transfer connection
    */

    Close:
    cpt_acl.acl_token = cpt_acm.acm_token;
    cpt_acl.acl_optns = ACLOPT_ORDER;
    t09fclos (&cpt_acl);
    if (cpt_acl.acl_rtncd != 0)
    {
    /* process and log error */
    }

    /*
    *  Orderly Release Listen connection
    */

    cpt_acl.acl_token = cpt_acm.acm_tlstn;
    cpt_acl.acl_optns = ACLOPT_ORDER;
    t09fclos (&cpt_acl);
    if (cpt_acl.acl_rtncd != 0)
    {
    /* process and log error */
    }

    /*
    *  Terminate Transaction
    */

    End:
    EXEC CICS RETURN;
}
```

024040-200100

CONNECT

This service provides a client facility for use by an application program. The CONNECT service establishes a session with the local transport provider, and then actively connects to a server. When connection is established with a server, the CONNECT service returns control to the calling program. Information related to the connection is updated and returned within the ACM.

To invoke the CONNECT service, a user application is required to first build an ACM and then to issue a call to the CONNECT routine. The minimum information required by this service is version number, server host address, and well-known port. Optional information related to data transfer buffering, CPT statistics and tracing, and subtask initialization can be specified.

This table describes the arguments for the CONNECT service:

INCLUDE STATEMENT	STRUCT NAME	SIZE	CREATED BY
t09ksacm.h	acm_stru	676 (X'2A4')	User application

Structure

This is the structure of the CONNECT service in C language:

```
#ifndef __t09ksacm__
#define __t09ksacm__
 /* t09ksacm.h --  Declare CPT-dependent structures    */

 /*     ACCCSECT UPDTID=A1000000                        */

 /*                                                     */
 /* Define Connection Management (ACM) Structure        */
 /*                                                     */

typedef struct
     {
    short acm_vers     ; /* ACM block version number  */
    short acm_func     ; /* Request function type     */
    int  *acm_token    ; /* Data transfer token       */
    void *acm_rsvd1    ; /* reserved (in use)         */
    int   acm_rsvd2    ; /* reserved (in use)         */
    int   acm_rtncd    ; /* Return code               */
    int   acm_dgncd    ; /* Diagnostic code           */
   char  acm_sfill [3] ; /* Reserved statistics bytes */
    char  acm_stats    ; /* Statistics options        */
   char  acm_tfill [2] ; /* Reserved trace bytes2     */
    char  acm_trac2    ; /* Trace options byte 2      */
    char  acm_trac1    ; /* Trace options byte 1      */
    int   acm_qsend    ; /* Send queue size           */
    int   acm_msend    ; /* Send buffer size          */
    int   acm_qrecv    ; /* Receive queue size        */
    int   acm_mrecv    ; /* Receive buffer size       */
    int  *acm_tlstn    ; /* Listen token              */
    int   acm_rsvd3    ; /* reserved (in use)         */
    char  acm_trnid [5] ; /* Transaction id           */
```

200801-024040-200100

```
    char  acm_rsvd4     ;  /* reserved                */
    short acm_rsvd5     ;  /* reserved                */
    short acm_lport     ;  /* Transport local port no.  */
    short acm_rport     ;  /* Transport remote port no. */
    char  acm_srvce[37] ;  /* Transport service name    */
    char  acm_rsvd6     ;  /* reserved                */
    short acm_optns     ;  /* Special options         */
    int   acm_laddr     ;  /* IP local host address   */
    int   acm_raddr     ;  /* IP remote host address  */
    char  acm_lname[256];  /* IP local host name      */
    char  acm_rname[256];  /* IP remote host name     */
    short acm_rsvd7     ;  /* reserved                */
    short acm_rsvd8     ;  /* reserved                */
    int   acm_timeo     ;  /* Timeout                 */
    int   acm_rsvd9     ;  /* reserved                */
    } acm_stru;

    /*-------------------------------------------*/
     /*       CONTROL BLOCK VERSION NUMBER       */
    /*-------------------------------------------*/
#define ACM_VERSN      2

    /*-------------------------------------------*/
     /*            STATISTICS OPTIONS           */
    /*-------------------------------------------*/
#define ACMSTATS_CONN    0x01 /* CONNECTION STATISTICS */
#define ACMSTATS_TERM    0x02 /* TERMINATION STATISTICS*/

    /*-------------------------------------------*/
     /*            TRACE LOG REQUESTS           */
    /*-------------------------------------------*/
#define ACMTRAC1_NTRY   0x01   /*    ENTRY POINTS    */
#define ACMTRAC1_ARGS   0x02   /*     ARGUMENTS      */
#define ACMTRAC1_RECV   0x04   /*       TRECV        */
#define ACMTRAC1_SEND   0x08   /*       TSEND        */
#define ACMTRAC1_TERM   0x10   /*    TERMINATION     */
#define ACMTRAC1_PASS   0x20   /*     GIVE/TAKE      */
#define ACMTRAC1_CLSE   0x40   /*      CLOSE         */
#define ACMTRAC1_TERR   0x80   /*     TPLERRORS      */
#define ACMTRAC2_TOKN   0x01   /*      TOKENS        */
#define ACMTRAC2_TPL    0x02   /*        TPL         */
#define ACMTRAC2_RLSE   0x04   /*      RELEASE       */
#define ACMTRAC2_STOR   0x08   /*      STORAGE       */

    /*-------------------------------------------*/
     /*            TRACE LOG REQUESTS           */
    /*-------------------------------------------*/
#define ACMOPTN1_SYNC   0x01   /*  listen syncpoint  */
#define ACMOPTN1_LTRAN  0x02   /* dynamic transaction */
#define ACMOPTN1_NODNR  0x01   /* no dnr service calls */

    /*-------------------------------------------*/
     /*       CPT CONNECTION MANAGEMENT SERVICES  */
    /*-------------------------------------------*/
#if defined (IBMC)
 #pragma linkage(t09fconn,OS)
 #pragma linkage(t09flstn,OS)
#else
 #define t09fconn __asm_t09fcon
```

```
 #define t09flstn __asm_t09flst
#endif

void t09fconn (), t09flstn ();

#endif
```

PARAMETER	DESCRIPTION
acm_vers	Version Indicates the CPT version number of the argument list used by the calling program. This required field must be set to ACM_VERSN for this release of CPT. Default: None
acm_func	Function code Indicates the function or callable service ID requested by the application program. This field is not set by the application, but rather is initialized by the TRUE interface stub program. Default: None (generated by service stub)
acm_token	TCP connection token Specifies the token is created and returned by the CONNECT service. This token is used for all subsequent service calls for the client connection. Default: 0 (token returned)
acm_rtncd	Return code Indicates the return code set by the CONNECT service. This value indicates the success or failure of the service. Default: 0
acm_dgncd	Diagnostic code Indicates the diagnostic code received by the CONNECT service for a transport provider request. A detailed explanation of this value is in the transport provider's ***API Programmer's Reference Guid***e. Default: 0
acm_stats	acmstats_conn | acmstats_term Specifies statistics logging options for the application program. The facility can be used for debugging and tuning during development. ◆ acmstats_conn – Specifies that a message(s) be generated on establishing either a listen service or a data transfer connection. These messages are generated by the CPT LISTEN and CONNECT services. The message numbers associated with this option are CPT802I, CPT803I, CPT811I, and CPT814I. ◆ acmstats_term – Specifies that a message(s) be generated on terminating an established connection. These messages are generated by the CPT CLOSE service. The message numbers associated with this option are CPT807I, CPT808I, CPT809I, CPT812I, CPT815I, CPT816I, CPT817I, and CPT818I. ◆ Default: 0 (no statistics logging)

200801-024040-200100

PARAMETER	DESCRIPTION
acm_trac1	Specifies trace logging options for the application program. The facility can be used for debugging during development. ◆ ACMTRAC1_NTRY – Specifies that a message be generated on entry to a CPT service routine. The message is generated by all CPT service routines. The message number associated with this option is CPT901I. ◆ ACMTRAC1_ARGS – Specifies that a hex dump of the caller's arguments be generated on entry and exit of an CPT service routine. Messages are generated by the corresponding service routine. Message numbers associated with this option are CPT902I, CPT903I, CPT904I, CPT911I, CPT912I, CPT921I,CPT922I, CPT924I, CPT925I, CPT930I, and CPT932I. ◆ ACMTRAC1_RECV – Specifies that a hex dump of the transport provider's input (RECEIVE) data be logged. Messages are generated by the CPT RECV service. The message number associated with this option is CPT913I. ◆ ACMTRAC1_SEND – Specifies that a hex dump of the transport provider's output (SEND) data be logged. Messages are generated by the CPT SEND service. The message number associated with this option is CPT914I. ◆ ACMTRAC1_TERM – Specifies that a message be generated on termination of an CPT service routine. Messages are generated by all CPT service routines. The message number associated with this option is CPT908I. ◆ ACMTRAC1_PASS – Specifies that a hex dump of resources related to a passed connection be logged. Messages are generated by the CPT GIVE and TAKE services. The message numbers associated with this option are CPT919I and CPT920I. ◆ ACMTRAC1_CLSE – Specifies that a hex dump of resources related to a CLOSE processing of a connection be logged. Message are generated by the CPT CLOSE service. The message number associated with this option is CPT906I. ◆ ACMTRAC1_TERR – Specifies that a hex dump of a transport provider API parameter list that fails successful completion be logged. The message number associated with this option is CPT400I.

20 024040-200100

PARAMETER	DESCRIPTION
acm_trac2	◆ ACMTRAC2_TOKN – Specifies that a hex dump of the connection token be logged. Messages are generated by all services routines on entry. The message number associated with this option is CPT909I. ◆ ACMTRAC2_TPL – Specifies that a hex dump of a transport provider API parameter list be logged. The message numbers associated with this option are CPT917I and CPT935I. ◆ ACMTRAC2_RLSE – Specifies that a message indicating a transport provider release indication be logged. Messages are generated by various CPT service routines. The message number associated with this option is CPT414I. ◆ ACMTRAC2_STOR – Specifies that a hex dump of storage management arguments be logged. Messages are generated by various CPT service routines. The message number associated with this option is CPT928I and CPT929I. ◆ ACMTRAC2_CLTD – Trace transient data writes from the LISTEN service (used with the ACMOPTN1_LTRAN listen start transaction option). The message number associated with this option is CPT918I. Default: 0 (no trace logging)
acm_qsend	API send queue size Specifies the maximum number of uncompleted SEND requests that can be queued by the application to the transport provider (API). This value lets applications control output processing and can affect throughput rates. The value is negotiated with the transport provider and may be modified by the transport provider. The total allocation for output processing is the product of the SEND queue and buffer size values and cannot exceed 61K. Default: 4
acm_msend	API send buffer size Specifies the maximum number of user data bytes that can be transferred by the application in a single SEND request to the transport provider (API). This value lets applications control output processing and can affect throughput rates. The value is negotiated with and may be modified by the transport provider. The total allocation for output processing is the product of the SEND queue and buffer size values and cannot exceed 61K. Default: 1024
acm_qrecv	API receive queue size Specifies the maximum number of uncompleted RECEIVE requests that can be queued by the application to the transport provider (API). This value lets applications control input processing and can affect throughput rates. The value is negotiated with and may be modified by the transport provider. The total allocation for input processing is the product of the RECEIVE queue and buffer size values and cannot exceed 61K. Default: 4
acm_mrecv	API receive buffer size Specifies the maximum number of user data bytes that can be transferred by the application, in a single RECEIVE request, to the transport provider (API). This value applications control input processing and can affect throughput rates. The value is negotiated with and may be modified by the transport provider. The total allocation for input processing is the product of the RECEIVE queue and buffer size values and cannot exceed 61K. Default: 1024

200801-024040-200100

PARAMETER	DESCRIPTION
acm_tlstn	Listen service token This field is not used by the CONNECT service. The value in this field is not validated nor is it modified. Default: None
acm_trnid	Listen start transaction ID This field is not used by the CONNECT service. The value in this field is not validated nor is it modified. Default: None
acm_lport	Listen well-known service port This value represents the TCP port on the local host that was assigned to the client application by TCP. It is returned to the caller of the CONNECT service. This field is an unsigned positive integer with a maximum value of 65,534. Default: None
acm_rport	Remote well-known service port Indicates the remote transport layer address or port. This value represents the TCP port on the remote host to which the client application is trying to connect. It must be filled in by the calling client application unless the acm_srvce field is specified. If acm_srvce is specified, acm_rport will be filled in with the resolved remote port number before returning to the caller of the CONNECT service. This field is an unsigned positive integer with a maximum value of 65,534. Default: None
acm_srvce	Transport layer service name Indicates the local transport layer service name. This string is mapped into a transport layer address or port, through internal Domain Name Resolution (DNR). The resolved value is the well-known port to which the CONNECT service attempts to establish a connection. This field has a maximum length of 36 bytes. This field is optional and is not modified by the CONNECT service. Default: None
acm_optns	Special options This field specifies TCP connection initialization options. ◆ ACMOPTN1_NODNR – Skip internal DNR calls to resolve and return the remote IP address into an IP name in the acm_rname field. If an application is designed such that TCP connection establishment and release happens frequently, this option can save processing time. ◆ ACMOPTN1_LTRAN – This option is for the LISTEN service and is not validated or modified by the CONNECT service. ◆ ACMOPTN1_SYNC – This option is for the LISTEN service and is not validated or modified by the CONNECT service. Default: None
acm_laddr	Local IP host address Indicates the local host internet address. This field is an unsigned four-byte integer value. The local host internet address is updated when a server connection is established, and is returned to the caller. Default: None

024040-200100

PARAMETER	DESCRIPTION
acm_raddr	Remote IP host address Indicates the remote host internet address. Either this field or the remote host name (acm_rname) field must be specified. This field is an unsigned four-byte integer value. The remote host internet address is updated when a server connection is established, and is returned to the caller. Default: None
acm_lname	Local IP host name Indicates the local host internet name. This field is a 255-byte character string that is padded with blanks. The local host internet name is updated when a server connection is established, and is returned to the caller. Default: None
acm_rname	Remote IP host name Indicates the remote host internet name. Either this value or the remote IP address (acm_raddr) field must be specified. This field is a 255-byte character string that is padded with blanks. The remote host internet name is updated when a server connection is established, and is returned to the caller. Default: None
acm_timeo	Maximum time to wait for client-specified server options. This field is optionally used by the LISTEN service and is not validated or modified by the CONNECT service. Default: 1 second

200801-024040-200100

Network Considerations

The ACM is a common data structure used for both client and server connection initialization. There are common and unique values specified for a particular service request.

This table describes network programming considerations for C API programming.

NAME	SERVER CONDITIONS FOR LISTEN	CLIENT CONDITIONS FOR CONNECT
acm_tlstn	Listen token returned to user application.	
acm_trnid	Listen START transaction ID.	
acm_lport	Local server or listening transport provider well-known port selected by user application.	Local assigned transport provider port returned to user application.
acm_rport	Remote client transport provider port returned to user application.	Remote server transport provider well-known port selected by user application.
acm_raddr	Remote IP host address returned to user application.	Remote IP host address selected or returned to user application
acm_srvce	Local server transport provider service name selected by user application.	Remote server transport provider service name selected by user application.
acm_lname	Local IP host name returned to user application.	Remote IP host name returned to user application.
acm_rname	Remote IP host name returned to user application	Remote IP host name selected or returned to user application.
acm_timeo	Maximum time to wait for client specified server options.	

Completion Information

The CONNECT service completes normally when a connection with a server is established. The CONNECT service initializes the client environment with the transport provider (API), actively contacts a server, and updates connection information within the ACM. Establishing a client connection is represented by storage and is referred to as the token. Successfully establishing a connection causes the ACM to be updated with information related to the connection.

The ACM is updated with information related to the new connection. The local and remote port, IP address, and host names are resolved. Negotiated transport provider SEND and RECEIVE buffering values are returned. Check the ACM return code (acm_rtncd) checked to determine the success or failure of the LISTEN service. A zero (0) return code indicates a successful connection.

The return and diagnostic codes should be interpreted by the application to determine the reason for failure. Errors indicating CPT, the transport provider (API), or CICS termination are minor. Interrogate errors for level of severity.

Return Codes

The CONNECT service return and diagnostic codes indicate the result of the execution. These values are in the acm_rtncd and acm_dgncd within the ACM. The diagnostic code generally indicates the transport provider return code. Read **Appendix C – MESSAGES AND CODES** in the ***CICS Programmer's Toolkit Installation and Administration Guide*** for the return code cross reference table.

This table describes the CONNECT service return codes:

RETURN	DIAGNOSTIC CODE	DESCRIPTION
CPTIRCOK		Successful.
CPTEVERS		Control block version number is not supported.
CPTECONN	Yes	Requested host/service/port is not found.
CPTEPROT		Specified protocol is not supported.
CPTENAPI	Yes	Transport provider API is not available.
CPTETERM	Yes	Environment is being terminated.
CPTERLSE	Yes	Release indication.
CPTEDISC	Yes	Disconnect indication.
CPTEPRGE	Yes	CPT Interface terminating.
CPTEINTG	Yes	Transport provider API integrity error.
CPTEENVR	Yes	Transport provider API environment error.
CPTEFRMT	Yes	Transport provider API format error.
CPTEPROC	Yes	Transport provider API procedure error.
CPTABEND		Abnormal exception occurred.
CPTEOTHR	Yes	An undefined exception occurred.

200801-024040-200100

Usage Information

The CONNECT service lets user-written application programs implement TCP/IP client facilities. The CONNECT service has a generalized parameter list referred to as the ACM. The ACM describes the application's communications requirements as well as information related to established connections. On completion, the ACM contains fields initialized by both a user application and by the CONNECT service.

There are required and optional fields initialized by a user or calling application. The ACM version number is required. The server must be identified by the calling program. The server is specified by selecting the remote IP address (acm_raddr) or host name (acm_rname) fields, and the remote port (acm_rport) or service name (acm_srvce). The selection of port or service name defines the server well-known port address. Optional fields control data transfer buffering, statistics, tracing, and subtask initialization.

On completion of the CONNECT service, the ACM contains information related to the established connection. A token that identifies the connection is returned in the ACM, and must be used in all subsequent requests that refer to the connection. The user application program should make no assumptions regarding the format of a token, other than that it is an unsigned, full word value. Information related to the negotiated buffer values, host names, host addresses, and transport provider addresses are returned in the ACM.

The version number (acm_vers) indicates the CPT release level in which this user application program is written. This required field must be set to ACM_VERSN and is validated by the CLOSE service before processing the request.

The function code (acm_func) indicates the CPT callable service ID. The field is initialized by the CPT service stub program and has little value to the application except for dump analysis. The function code can identify and map an argument with the error or trace logs, and dump analysis.

The remote IP address (acm_raddr) or remote host name (acm_rname) is required. These fields identify the host to which the CONNECT service initiates a connection request. The IP address has precedence over host name. This implies that the host name field is only used if an IP address is not specified.

The transport provider port number (acm_rport) or service name (acm_srvce) is required. These fields identify the well-known port to which the CONNECT service initiates a connection request. The port number has precedence over service name. This implies that the service name field is only used if a port number is not specified.

User application programs have the ability to control CPT and transport provider data transfer buffering. The acm_qsend, acm_msend, acm_qrecv, and acm_mrecv specify the number and size of buffers allocated. The SEND and RECEIVE buffers are allocated on initial entry into either the SEND or RECEIVE service. The corresponding values used by the SEND and RECEIVE services are independent of each other. The SEND service multiplies the queue and buffer values to determine output storage requirements. The RECEIVE service performs a similar function to determine input storage requirements. The product of the queue and buffer values cannot exceed 61,440. CPT requires

some additional storage to manage these buffers. This extra storage is included in the allocation.

The SEND service uses the acm_msend value, while the RECEIVE service uses the acm_mrecv value. These values indicate the maximum number of uncompleted SEND or RECEIVE requests that can be queued by the application to the transport provider. The CPT data transfer services schedule asynchronous transport provider or API requests on the caller's behalf. These API requests must be completed before they can be used again. If the queue values are too small, the CPT data transfer service may block the caller's request and schedule a WAIT command within the service routines. If the queue values are too large, the user application may be wasting storage.

The SEND service uses the acm_msend value. The RECEIVE service uses the acm_mrecv. These values indicate the maximum number of user data bytes that can be transferred by the application in a single SEND or RECEIVE request to the transport provider. The user application is not limited to these values within the data transfer services. However, it is important to note that multiple transport provider or API requests will be issued to complete the caller's request. Information on queue and buffer size is in the descriptions of **RECEIVE** on page 3-56 and **SEND** on page 3-65.

Initially, the tuning of data transfer storage may not be a concern; however, the ability to control storage allocation can prove beneficial to the application or CICS region. Additionally, queue size can increase data transfer throughput. Consider enabling the statistics option to gather CPT statistical information, which can be used to set the SEND or RECEIVE queue and buffer size values.

The CONNECT service can modify data transfer buffer allocation values. These values are negotiated with the transport provider and, depending on the site configuration, can be reduced. Any application dependent on these values should check them on return. These values are generally not modified when giving reasonable numbers. However, it is advisable to check with the site administrator for maximum values for the API transport services.

A number of arguments are not set by the calling application, but are returned to the caller. These values represent information related to the client connection and can be used by the application. The local port, host name, and IP address are returned as well as the client's corresponding values.

2008n1-024040-200100

Example:

This example establishes a connection to a LOOPBACK host on port 7. The token is loaded from the ACM and used by all of the following CPT service requests. The `acl_rtncd` is checked to determine `CONNECT` service request completion status.

```
#include  <t09ksacm.h>
#include  <t09ksrcs.h>

void main()
{
   struct acm_stru
      cpt_acm = {2, 0, 0, NULL, NULL, 0, 0, 0, 0, 0, 0, 0, 0,
      NULL, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};

   /*
      *  CPT Connection Management Service request
   */

   cpt_acm.acm_rport = 7;
   memcpy(cpt_acm.acm_rname,"LOOPBACK",8);
   cpt_acm.acm_proto = ACMPROTO_TCP;
   t09fconn (&cpt_acm);
   if (cpt_acm.acm_rtncd != 0)
   {
   /* process CPT CONNECT service error and terminate transaction */
   }

   cpt_adt.adt_token = cpt_acm.acm_token;

   while (data)
   {
      /*
      *  Application and CPT data transfer (SEND/RECEIVE) processing
      */
   }

   /*
   *  CPT Connection Release
   */

   /*
   *  Terminate Transaction
   */

   End:
   EXEC CICS RETURN;
}
```


024040-200100

Example:

This example establishes a connection to a LOCALHOST host and selects the server service name of ECHO selected. The token is loaded from the ACM and used by all of the following CPT service requests. The acl_rtncd is checked to determine CONNECT service request completion status.

```
#include  <t09ksacm.h>
#include  <t09ksrcs.h>

void main()
{
    struct acm_stru
        cpt_acm = {2, 0, 0, NULL, NULL, 0, 0, 0, 0, 0, 0, 0, 0,
        NULL, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};

    /*
    *  CPT Connection Management Service request
    */
    memcpy(cpt_acm.acm_srvce, "ECHO", 4);
    memcpy(cpt_acm.acm_rname, "LOOPBACK", 8);
    cpt_acm.acm_proto = ACMPROTO_TCP;
    t09fconn (&cpt_acm);
    if (cpt_acm.acm_rtncd != 0)
    {
        /* process CPT CONNECT service error and terminate transaction */
    }

    cpt_adt.adt_token = cpt_acm.acm_token;

    while (data)
    {
        /*
        *  Application and CPT data transfer (SEND/RECEIVE) processing
        */
    }

    /*
    *  CPT Connection Release
    */

    /*
    *  Terminate Transaction
    */

    End:
    EXEC CICS RETURN;
}
```

200801-024040-200100

GIVE

The GIVE service releases ownership of a connection and associated internal CPT resources. The GIVE service is optional and does not affect an active connection, nor does it issue any transport provider requests. This service affects CPT TRUE management routines, scheduled on the user's behalf during task termination.

To invoke the GIVE service, a user application is required to first build an Argument for Facility Management (AFM) and then to issue a call to the GIVE routine. The only valid and required arguments are the AFM version number and the connection token. On completion, a return code indicates the success or failure of the request.

This table describes the arguments for the GIVE service:

INCLUDE STATEMENT	STRUCT NAME	SIZE	CREATED BY
t09ksafm.h	afm_stru	34 (X'22')	user application

Structure

This is the structure of the GIVE service in C language:

```
#ifndef __t09ksafm__
#define __t09ksafm__
 /* t09ksafm.h --  Declare CPT-dependent structures     */

 /*     ACCCSECT UPDTID=A1000000                         */

 /*                                                      */
 /* Define Facility Management (AFM) Structure           */
 /*                                                      */

typedef struct
      {
     short afm_vers     ; /* AFM block version number  */
     short afm_func     ; /* Request function type     */
     int  *afm_token    ; /* Data Transfer token       */
     void *afm_buffa    ; /* Reserved                  */
     int   afm_buffl    ; /* Reserved                  */
     int   afm_rtncd    ; /* Return code               */
     int   afm_dgncd    ; /* Diagnostic code           */
     int   afm_rsvd1    ; /* reserved (in use)         */
     short afm_rsvd2    ; /* reserved (in use)         */
      } afm_stru;

     /*-------------------------------------------*/
       /*       CONTROL BLOCK VERSION NUMBER       */
     /*-------------------------------------------*/
#define AFM_VERSN       2

     /*-------------------------------------------*/
       /*       CPT FACILITY MANAGEMENT SERVICE    */
     /*-------------------------------------------*/
 #if defined (IBMC)
  #pragma linkage(t09fgive,OS)
  #pragma linkage(t09ftake,OS)
 #else
```

```
 #define t09fgive __asm_t09fgiv
 #define t09ftake __asm_t09ftak
#endif

void t09fgive (), t09ftake ();

#endif
```

PARAMETER	DEFINITION
afm_vers	Version Indicates the CPT version number of the argument list used by the calling program. This required field must be set to AFM_VERSN for this release of CPT. Default: None
afm_func	Function code Indicates the function or callable service ID requested by the application program. This field is not set by the application, but rather is initialized by the TRUE interface stub program. Default: None
afm_token	Connection token Specifies a token that represents a TCP connection, a TCP listening endpoint, or a UDP endpoint. A token is created by the TCP connection initiation routines or by the UDP data transfer and endpoint creation routines. The token is required. Default: 0
afm_rtncd	Return code Indicates the return code set by the GIVE service. This value indicates the success or failure of the service. Default: 0
afm_dgncd	Diagnostic code Indicates the diagnostic code received by the GIVE service for a transport provider request and is not set by the GIVE service. The GIVE service does not issue transport provider requests; hence, it never sets the diagnostic code. Default: 0

200801-024040-200100

Completion Information

The GIVE service completes normally when all resources associated with this connection are processed.

On normal return to the application program, the general return code in afm_rtncd is set to zero (CPTIRCOK). The diagnostic code in afm_dgncd is always zero.

If the GIVE service completes abnormally, some resources associated with this connection cannot be successfully transferred from one task to another. The general return code in afm_rtncd and the diagnostic code in afm_dgncd indicate the nature of the failure. The diagnostic code (afm_dgncd) is not used by the GIVE service and no information is returned.

Return Codes

The GIVE service return and diagnostic codes indicate the result of the execution. These values are in the afm_rtncd and afm_dgncd within the AFM. Read **Appendix C – MESSAGES AND CODES** in the ***CICS Programmer's Toolkit Installation and Administration Guide*** for the return code cross reference table.

This table describes the GIVE service return codes:

RETURN	DESCRIPTION
CPTIRCOK	Successful.
CPTEVERS	Control block version number is not supported.
CPTETOKN	Specified token is not valid.
CPTENAPI	Transport provider API is not available.
CPTABEND	Abnormal exception occurred.
CPTEOTHR	An undefined exception occurred.

224040-200100

Usage Information

The GIVE service releases ownership of a connection. This service is non-blocking and does not affect any pending transport provider data transfer requests. Disassociating resources from a task lets the CPT properly manage resources during task termination. This ability to GIVE and TAKE ownership of connections offers the user a range of programming options, while still providing CPT with resource management capabilities.

The GIVE service utilizes the AFM. The GIVE service requires the application to set the AFM version number and token fields. No other fields are referenced.

It is important to understand that when a connection is established there are internal CPT resources associated with that connection. CPT is responsible for proper clean-up of those resources on task or transaction termination. These resources include storage allocated by CPT, the API, and the transport provider storage.

A server application is a good example of how the GIVE service benefits a user application. A listening task issues the GIVE service and starts a new transaction to handle data transfer. The data transfer transaction then takes the connection. This sequence prevents a connection from being closed (implicitly by the CPT task termination exit) if the server application terminates. However, if the data transfer transaction is terminated without issuing an explicit close (CPT CLOSE service) an implicit close is scheduled, and resource management is handled by the CPT task termination exit.

The version number (afm_vers) indicates the CPT release level in which this user application program is written. This required field must be set to AFM_VERSN and is validated by the GIVE service before processing the request.

The function code (afm_func) indicates the CPT callable service ID. The field is not initialized by a user application program and has little value to the application except for dump analysis. The function code identifies and maps an argument list with the error or trace log and dump analysis.

The token (afm_token) indicates the connection and internal resources to be processed by the GIVE service. This is a required field and is validated by the GIVE service before processing the request.

The afm_optcd field specifies GIVE service processing control options, and provides a mechanism for event notification on return to the application program. Currently, this field is not used by application programs.

200Rn1-024040-200100

Example:

In this example a LISTEN service establishes a client connection and then issues the GIVE service before starting a new transaction to handle data processing. The ACM is provided to the data processing transaction. The afm_rtncd is checked to determine GIVE service request completion status.

```
#include  <t09ksacm.h>
#include  <t09ksafm.h>
#include  <t09ksrcs.h>

void main()
{
    struct acm_stru
        cpt_acm = {2, 0, 0, NULL, NULL, 0, 0, 0, 0, 0, 0, 0, 0,
        NULL, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
    struct afm_stru
        cpt_afm = {2, 0, 0, NULL, NULL, 0, 0, 0, 0};

    while (true)
    {
        /*
            * CPT LISTEN Connection Management service request
        */

        if (cpt_acm.acm_rtncd != 0)
        {
        /* process CPT LISTEN service error and terminate transaction */
        }
        cpt_afm.afm_token = cpt_acm.acm_token;
        t09fgive (&cpt_afm);
        if (cpt_afm.afm_rtncd != 0)
        {
            /*
            *  process CPT GIVE service error, release connection
            *  and terminate transaction
            */
        }
        EXEC CICS START TRANSID(trans-id) FROM(cpt_acm);
    }

    /*
    *  CPT Connection Release
    */

    /*
    *  Terminate Transaction
    */

    End:
    EXEC CICS RETURN;
}
```

Example:

In this example a LISTEN service establishes a client connection and then issues the GIVE service before starting a new transaction to handle data processing. The token is provided to the data processing transaction. The afm_rtncd is checked to determine GIVE service request completion status.

```
#include  <t09ksacm.h>
#include  <t09ksafm.h>
#include  <t09ksrcs.h>

void main()
{
    struct acm_stru
        cpt_acm = {2, 0, 0, NULL, NULL, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        NULL, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
    struct afm_stru
        cpt_afm = {2, 0, 0, NULL, NULL, 0, 0, 0, 0};

    while (true)
    {
        /*
        *  CPT LISTEN Connection Management service request
        */

    if (cpt_acm.acm_rtncd != 0)
    {
        /* process CPT LISTEN service error and terminate transaction */
        }
        cpt_afm.afm_token = cpt_acm.acm_token;
        t09fgive (&cpt_afm);
        if (cpt_afm.afm_rtncd != 0)
        {
            /*
            * process CPT GIVE service error, release connection
            * terminate transaction
            */
        }
        EXEC CICS START TRANSID(trans-id) FROM(cpt_afm.afm_token);
    }

    /*
    *  CPT Connection Release
    */

    /*
    *  Terminate Transaction
    */

    End:
    EXEC CICS RETURN;
}
```

200801-024040-200100

LISTEN

This service provides a server facility for use by an application program. The LISTEN service establishes a session with the local transport provider, passively listens for connection requests, then accepts new connections. When connection with a client is established, the LISTEN service either returns control to the calling program or starts a defined transaction. Information related to the connection is updated and returned within the ACM.

To invoke the LISTEN service, a user application is required to first build an ACM, then issue a call to the LISTEN routine. The minimum information required by this service is version number and either the local transport provider port or service name. Optional information related to data transfer buffering, CPT statistics and tracing, and subtask initialization can be specified. Completion of a LISTEN service depends on options selected within the ACM.

This table describes the LISTEN service arguments:

INCLUDE STATEMENT	STRUCT NAME	SIZE	CREATED BY
t09ksacm.h	acm_stru	676 (X'2A4')	User application or common area obtained by a RETRIEVE command from a started transaction.

Structure

This is the structure of the LISTEN service in C language:

```
#ifndef __t09ksacm__
#define __t09ksacm__
 /* t09ksacm.h --  Declare CPT-dependent structures    */

 /*     ACCCSECT UPDTID=A1000000                       */

 /*                                                    */
 /*  Define Connection Management (ACM) Structure      */
 /*                                                    */

typedef struct
     {
    short acm_vers     ;  /* ACM block version number  */
    short acm_func     ;  /* Request function type     */
    int  *acm_token    ;  /* Data transfer token       */
    void *acm_rsvd1    ;  /* reserved (in use)         */
    int   acm_rsvd2    ;  /* reserved (in use)         */
    int   acm_rtncd    ;  /* Return code               */
    int   acm_dgncd    ;  /* Diagnostic code           */
   char  acm_sfill [3] ;  /* Reserved statistics bytes */
    char  acm_stats    ;  /* Statistics options        */
   char  acm_tfill [2] ;  /* Reserved trace bytes2     */
    char  acm_trac2    ;  /* Trace options byte 2      */
    char  acm_trac1    ;  /* Trace options byte 1      */
    int   acm_qsend    ;  /* Send queue size           */
    int   acm_msend    ;  /* Send buffer size          */
```

```
    int   acm_qrecv    ; /* Receive queue size       */
    int   acm_mrecv    ; /* Receive buffer size      */
    int  *acm_tlstn    ; /* Listen token             */
    int   acm_rsvd3    ; /* reserved (in use)        */
    char  acm_trnid [5] ; /* Transaction id          */
    char  acm_rsvd4    ; /* reserved                 */
    short acm_rsvd5    ; /* reserved                 */
    short acm_lport    ; /* Transport local port no. */
    short acm_rport    ; /* Transport remote port no. */
    char  acm_srvce[37] ; /* Transport service name  */
    char  acm_rsvd6    ; /* reserved                 */
    short acm_optns    ; /* Special options          */
    int   acm_laddr    ; /* IP local host address    */
    int   acm_raddr    ; /* IP remote host address   */
    char  acm_lname[256]; /* IP local host name       */
    char  acm_rname[256]; /* IP remote host name      */
    short acm_rsvd7    ; /* reserved                 */
    short acm_rsvd8    ; /* reserved                 */
    int   acm_timeo    ; /* Timeout                  */
    int   acm_rsvd9    ; /* reserved                 */
    } acm_stru;

    /*-------------------------------------------*/
     /*       CONTROL BLOCK VERSION NUMBER       */
    /*-------------------------------------------*/
#define ACM_VERSN      2

    /*-------------------------------------------*/
     /*            STATISTICS OPTIONS            */
    /*-------------------------------------------*/
#define ACMSTATS_CONN    0x01 /* CONNECTION STATISTICS */
#define ACMSTATS_TERM    0x02 /* TERMINATION STATISTICS*/

    /*-------------------------------------------*/
     /*            TRACE LOG REQUESTS            */
    /*-------------------------------------------*/
#define ACMTRAC1_NTRY   0x01   /*    ENTRY POINTS    */
#define ACMTRAC1_ARGS   0x02   /*     ARGUMENTS      */
#define ACMTRAC1_RECV   0x04   /*       TRECV        */
#define ACMTRAC1_SEND   0x08   /*       TSEND        */
#define ACMTRAC1_TERM   0x10   /*    TERMINATION     */
#define ACMTRAC1_PASS   0x20   /*     GIVE/TAKE      */
#define ACMTRAC1_CLSE   0x40   /*      CLOSE         */
#define ACMTRAC1_TERR   0x80   /*     TPLERRORS      */
#define ACMTRAC2_TOKN   0x01   /*      TOKENS        */
#define ACMTRAC2_TPL    0x02   /*        TPL         */
#define ACMTRAC2_RLSE   0x04   /*      RELEASE       */
#define ACMTRAC2_STOR   0x08   /*      STORAGE       */

    /*-------------------------------------------*/
     /*            TRACE LOG REQUESTS            */
    /*-------------------------------------------*/
#define ACMOPTN1_SYNC   0x01   /*  listen syncpoint  */
#define ACMOPTN1_LTRAN  0x02   /* dynamic transaction */
#define ACMOPTN1_NODNR  0x01   /* no dnr service calls */

    /*-------------------------------------------*/
     /*       CPT CONNECTION MANAGEMENT SERVICES  */
    /*-------------------------------------------*/
```

200801-024040-200100

```
#if defined (IBMC)
 #pragma linkage(t09fconn,OS)
 #pragma linkage(t09flstn,OS)
#else
 #define t09fconn __asm_t09fcon
 #define t09flstn __asm_t09flst
#endif

void t09fconn (), t09flstn ();

#endif
```

PARAMETER	DESCRIPTION
acm_vers	Version Indicates the CPT version number of the argument list used by the calling program. This required field must be set to ACM_VERSN for this release of CPT. Default: None
acm_func	Function code Indicates the function or callable service ID requested by the application program. This field is not set by the application, but rather is initialized by the TRUE interface stub program. Default: None (generated by service stub)
acm_token	Data transfer token Specifies the token is created and returned by the LISTEN service. This token is used for all subsequent service calls for the client connection. Default: 0 (token returned)
acm_rtncd	Return code Indicates the return code set by the LISTEN service. This value indicates the success or failure of the service. Default: 0
acm_dgncd	Diagnostic code Indicates the diagnostic code received by the LISTEN service for a transport provider request. A detailed explanation of this value is in the transport provider's ***API Programmer's Reference Guide***. Default: 0
acm_stats	ACMSTATS_CONN | ACMSTATS_TERM Specifies statistics logging options for the application program. The facility can be used for debugging and tuning during development. ◆ ACMSTATS_CONN – Specifies that a message(s) be generated on establishing either a listen service or a data transfer connection. These messages are generated by the CPT LISTEN and CONNECT services. The message numbers associated with this option are CPT802I and CPT803I. ◆ ACMSTATS_TERM – Specifies that a message(s) be generated on terminating an established connection. These messages are generated by the CPT CLOSE service. The message numbers associated with this option are CPT807I, CPT808I, and CPT809I. Default: 0 (no statistics logging)

20 024040-200100

PARAMETER	DESCRIPTION
acm_trac1	Specifies trace logging options for the application program. The facility can be used for debugging during development. ◆ ACMTRAC1_NTRY – Specifies that a message be generated on entry to a CPT service routine. The message is generated by all CPT service routines. The message number associated with this option is CPT901I. ◆ ACMTRAC1_ARGS – Specifies that a hex dump of the caller's arguments be generated on entry and exit of an CPT service routine. Messages are generated by the corresponding service routine. Message numbers associated with this option are CPT902I, CPT903I, CPT904I, CPT911I, CPT912I, CPT921I,CPT922I, CPT924I, CPT925I, CPT930I, and CPT932I. ◆ ACMTRAC1_RECV – Specifies that a hex dump of the transport provider's input (RECEIVE) data be logged. Messages are generated by the CPT RECV service. The message number associated with this option is CPT913I. ◆ ACMTRAC1_SEND – Specifies that a hex dump of the transport provider's output (SEND) data be logged. Messages are generated by the CPT SEND service. The message number associated with this option is CPT914I. ◆ ACMTRAC1_TERM – Specifies that a message be generated on termination of an CPT service routine. Messages are generated by all CPT service routines. The message number associated with this option is CPT908I. ◆ ACMTRAC1_PASS – Specifies that a hex dump of resources related to a passed connection be logged. Messages are generated by the CPT GIVE and TAKE services. The message numbers associated with this option are CPT919I and CPT920I. ◆ ACMTRAC1_CLSE – Specifies that a hex dump of resources related to a CLOSE processing of a connection be logged. Message are generated by the CPT CLOSE service. The message number associated with this option is CPT906I. ◆ ACMTRAC1_TERR – Specifies that a hex dump of a transport provider API parameter list that fails successful completion be logged. The message number associated with this option is CPT400I.

200801-024040-200100

PARAMETER	DESCRIPTION
acm_trac2	◆ ACMTRAC2_TOKN – Specifies that a hex dump of the connection token be logged. Messages are generated by all services routines on entry. The message number associated with this option is CPT909I. ◆ ACMTRAC2_TPL – Specifies that a hex dump of a transport provider API parameter list be logged. The message numbers associated with this option are CPT917I and CPT935I. ◆ ACMTRAC2_RLSE – Specifies that a message indicating a transport provider release indication be logged. Messages are generated by various CPT service routines. The message number associated with this option is CPT414I. ◆ ACMTRAC2_STOR – Specifies that a hex dump of storage management arguments be logged. Messages are generated by various CPT service routines. The message number associated with this option is CPT928I and CPT929I. ◆ ACMTRAC2_CLTD – Trace transient data writes from the LISTEN service (used with the ACMOPTN1_LTRAN Client-Data option). The message number associated with this option is CPT918I. Default: 0 (no trace logging)
acm_qsend	API send queue size Specifies the maximum number of uncompleted SEND requests that can be queued by the application to the transport provider (API). This value lets applications control output processing and can affect throughput rates. The value is negotiated with the transport provider and may be modified by the transport provider. The total allocation for output processing is the product of the SEND queue and buffer size values and cannot exceed 61K. Default: 4
acm_msend	API send buffer size Specifies the maximum number of user data bytes that can be transferred by the application in a single SEND request to the transport provider (API). This value lets applications control output processing and can affect throughput rates. The value is negotiated with and may be modified by the transport provider. The total allocation for output processing is the product of the SEND queue and buffer size values and cannot exceed 61K. Default: 1024
acm_qrecv	API receive queue size Specifies the maximum number of uncompleted RECEIVE requests that can be queued by the application to the transport provider (API). This value lets applications control input processing and can affect throughput rates. The value is negotiated with and may be modified by the transport provider. The total allocation for input processing is the product of the RECEIVE queue and buffer size values and cannot exceed 61K. Default: 4
acm_mrecv	API receive buffer size Specifies the maximum number of user data bytes that can be transferred by the application, in a single RECEIVE request, to the transport provider (API). This value applications control input processing and can affect throughput rates. The value is negotiated with and may be modified by the transport provider. The total allocation for input processing is the product of the RECEIVE queue and buffer size values and cannot exceed 61K. Default: 1024

2024040-200100

PARAMETER	DESCRIPTION
acm_tlstn	Listen service token statistics Specifies the token used by the LISTEN service. This token is not available for data transfer. The only valid function that can be performed is a CLOSE request for long running active listeners. Generally, this value is not used by the application unless an explicit call to the CLOSE service is required. Read the description for ACMTOKEN (earlier in this table) for all other services. Default: 0 (token returned)
acm_trnid	Listen start transaction ID A four-byte character string that the LISTEN service starts on successful establishment of a new connection. If TRANSID is specified, the LISTEN server loops for new connections and does not return to the calling program until CICS, CPT, or transport provider (API) termination. This field is optional and is not modified by the listen service. This field should not be specified if the ACMOPTN1_LTRAN option and acm_timeo value are specified. Default: None
acm_lport	Listen well-known service port Indicates the local transport layer address or port. This value represents the well-known port on which a server application will listen for connection requests. Either this value or the transport layer service name (ACMSRVCE) must be specified. This field is an unsigned positive integer with a maximum value of 65,534. The value must be unique for each server application. Default: None
acm_rport	Remote well-known service port Indicates the remote transport layer address or port. This value represents the TCP port on the remote host to which the client application is trying to connect. It must be filled in by the calling client application unless the acm_srvce field is specified. If acm_srvce is specified, acm_rport will be filled in with the resolved remote port number before returning to the caller of the CONNECT service. This field is an unsigned positive integer with a maximum value of 65,534. Default: None
acm_srvce	Transport layer service name Indicates the local transport layer service name. This string is mapped into a transport layer address or port, through internal DNR. The resolved value is the well-known port to which the client application connects. This field has a maximum length of 36 bytes. The value must be unique for each server application. This field is optional and is not modified by the LISTEN service. Default: None
acm_optns	Specifies TCP connection initialization options ◆ ACMOPTN1_NODNR – Skip internal DNR calls to resolve and return the remote IP address into an IP name in the acm_rname field. If an application is designed such that TCP connection establishment and release happens frequently, this option can save processing time. ◆ ACMOPTN1_LTRAN – Client-Data Listener option. Specifies that the Listen call will receive the input datastream to determine the transaction ID to be started. See Client-Data Listener option for the required input formats. This option must be used with acm_timeo, and should not be used with acm_trnid. ◆ ACMOPTN1_SYNC – Listen Syncpoint option. Issues a CICS syncpoint before starting any transaction from the LISTEN service. Default: None

200801-024040-200100

PARAMETER	DESCRIPTION
acm_laddr	Local IP host address Indicates the local host internet address. This field is an unsigned four-byte integer value. The local host internet address is updated when a server connection is established, and is returned to the caller. Default: None
acm_raddr	Remote IP host address Indicates the remote host internet address. Either this field or the remote host name (acm_rname) field must be specified. This field is an unsigned four-byte integer value. The remote host internet address is updated when a server connection is established, and is returned to the caller. Default: None
acm_lname	Local IP host name Indicates the local host internet name. This field is a 255-byte character string that is padded with blanks. The local host internet name is updated when a server connection is established, and is returned to the caller. Default: None
acm_rname	Remote IP host name Indicates the remote host internet name. Either this value or the remote IP address (acm_raddr) field must be specified. This field is a 255-byte character string that is padded with blanks. The remote host internet name is updated when a server connection is established, and is returned to the caller. Default: None
acm_timeo	Client-Data Listener timeout values Specifies the maximum number of seconds that a Listener can wait to receive the client datastream when the ACMOPTN1_LTRAN option is specified. Default: 1 second

Network Considerations

The ACM is a common data structure used for both client and server connection initialization. There are common and unique values specified for a particular service request.

This table describes network programming considerations for C API programming:

NAME	SERVER CONDITIONS FOR LISTEN	CLIENT CONDITIONS FOR CONNECT
acm_tlstn	Listen token returned to user application.	
acm_trnid	Listen START transaction ID.	
acm_lport	Local server or listening transport provider well-known port selected by user application.	Local assigned transport provider port returned to user application.
acm_rport	Remote client transport provider port returned to user application.	Remote server transport provider well-known port selected by user application.
acm_raddr	Remote IP host address returned to user application.	Remote IP host address selected or returned to user application
acm_srvce	Local server transport provider service name selected by user application.	Remote server transport provider service name selected by user application.
acm_lname	Local IP host name returned to user application.	Remote IP host name returned to user application.
acm_rname	Remote IP host name returned to user application	Remote IP host name selected or returned to user application.
acm_timeo	Maximum time to wait for client specified server options.	

200801-024040-200100

Completion Information

Completion of a request to the LISTEN service depends on the arguments selected. If no transaction ID is specified, the LISTEN service returns control to the calling program when a connection with a client is established. The caller's argument list is updated with various information related to the new connection. If a transaction ID was specified, the LISTEN service does not return control to the calling program until a failure is detected. The caller's argument list is updated with information related to the new connection.

The LISTEN service initializes the server environment with the transport provider (API), waits for a connection request, establishes a connection with the client, and updates connection information within the ACM. Establishing a listening connection and a client connection are represented by storage and are referred to hereafter as tokens. Establishing a client connection causes the ACM to be updated with information relative to the connection. The information is returned to the user or is passed to the data processing transaction.

The server application contains two tokens representing endpoints to the transport provider. The first token (acm_token) represents the client connection and is used for data transfer. The other token (acm_tlstn) represents the listening connection. This listening connection can only be referenced within the CPT CLOSE service. This lets an explicit ability close a server or listening connection. All other CPT services performed with the LISTEN token fail with an invalid token. Implicit clean-up of the LISTEN token is provided by the TRUE interface. Therefore, an explicit call to the CLOSE service is not required.

When the LISTEN service is initiated without a transaction ID, control is returned to the calling program when a connection with a client is established. The caller's argument list is updated with information related to the new connection. The local and remote port, IP address, and host names are resolved. Negotiated transport provider SEND and RECEIVE buffering values are returned. The ACM return code (acm_rtncd) must be checked to determine the success or failure of LISTEN service. A zero (0) return code indicates a successful client connection is established.

When the LISTEN service is initiated with a transaction ID, it operates as a CICS long running task. The LISTEN service establishes client connections and starts a data processing transaction. The data processing transaction receives a copy of the connection management argument. The data transfer or client connection token is derived from the acm_token field. After the new transaction is initiated the LISTEN service continues waiting for new client connections. The LISTEN service continues to listen and start client connections until an error occurs.

The return and diagnostic codes should be interrogated to determine the reason for failure. Errors indicating CPT, the transport provider (API), or CICS termination are minor. Errors should be interrogated for level of severity.

20-024040-200100

Return Codes

The LISTEN service return and diagnostic codes indicate the result of the execution. These values are in the acm_rtncd and acm_dgncd within the connection management argument. The diagnostic code is optional and indicates the transport provider return code. Read **Appendix C – MESSAGES AND CODES** in the ***CICS Programmer's Toolkit Installation and Administration Guide*** for the return code cross reference table.

This table describes the LISTEN service return codes:

RETURN	DIAGNOSTIC CODE	DESCRIPTION
CPTIRCOK		Successful.
CPTEVERS		Control block version number is not supported.
CPTECONN	Yes	Requested host/service/port is not found.
CPTEPROT		Specified protocol is not supported.
CPTENAPI	Yes	Transport provider API is not available.
CPTETERM	Yes	Environment is being terminated.
CPTERLSE	Yes	Release indication.
CPTEDISC	Yes	Disconnect indication.
CPTEPRGE	Yes	CPT Interface terminating.
CPTEINTG	Yes	Transport provider API integrity error.
CPTEENVR	Yes	Transport provider API environment error.
CPTEFRMT	Yes	Transport provider API format error.
CPTEPROC	Yes	Transport provider API procedure error.
CPTABEND		Abnormal exception occurred.
CPTEOTHR	Yes	An undefined exception occurred.

200801-024040-200100

Usage Information

The LISTEN service lets user-written application programs implement TCP/IP server facilities. Server applications passively wait, then establish connections with single- or multi-thread support. The LISTEN service has a generalized parameter list that is referred to as the ACM. The ACM describes the application's communications requirements as well as information related to established connections. The ACM contains fields initialized by both a user application and by the LISTEN service, on completion.

There are required and optional fields initialized by a user or calling application. The ACM version number is required. The transport provider local port or service name is required. The selection of port or service name defines the server well-known port address. Optional fields control data transfer buffering, statistics, tracing, and subtask initialization.

When the LISTEN service completes or the data processing task executes, the ACM contains information related to the established connection. A token that identifies the connection is returned in the ACM, and must be used in all subsequent requests that refer to the connection. The application program should make no assumptions regarding the format of a token, other than it is an unsigned, full word value.

Information related to the negotiated buffer values, host names, host addresses, and transport provider addresses are returned in the ACM.

The version number (acm_vers) indicates the CPT release level in which this user application program is written. This required field must be set to ACM_VERSN and is validated by the LISTEN service before processing the request.

The function code (acm_func) indicates the CPT callable service ID. The field is initialized by the CPT service stub program. The function code identifies argument lists within the error or trace logs, and dump analysis.

The transport provider port number (acm_rport) or service name (acm_srvce) is required. These fields identify the well-known port to which a client initiates a connection request. The port number has precedence over service name. This implies that the service name field is only used if a port number is not specified.

The transaction ID field (acm_trnid) identifies a data processing task. This is an optional field that causes the LISTEN service to execute continuously. The LISTEN service starts a new transaction after a client connection is established, then waits for additional connection requests. An updated ACM is passed to the data processing task. Control is not returned to the calling program until an error occurs. The return code indicates the reason for the failure. Errors indicating the transport provider, CICS, or CPT termination are acceptable. Errors indicating port in use, API unavailable, or program checks should be investigated.

User application programs can control CPT and transport provider data transfer buffering. The acm_qsend, acm_msend, acm_qrecv, and acm_mrecv specify the number and size of buffers allocated. The SEND and RECEIVE buffers are allocated on initial entry into either the SEND or RECEIVE service. The corresponding values used by the SEND and RECEIVE services are independent

of each other. The SEND service multiplies the queue and buffer values to determine output storage requirements. The RECEIVE service performs a similar function to determine input storage requirements. The product of the queue and buffer values cannot exceed 61,440. CPT requires some additional storage to manage these buffers. This extra storage is included in the allocation.

The CPT SEND service uses the acm_qsend value, and the CPT RECEIVE service uses the acm_qrecv value. These values indicate the maximum number of uncompleted SEND or RECEIVE requests that can be queued by the application to the transport provider. The CPT data transfer services schedule asynchronous transport provider or API requests on the caller's behalf. These API requests must be completed before they can be used again. If the queue values are too small the CPT data transfer service may block the caller's request and schedule a wait within the service routines. If the queue values are too large the user application may be wasting storage.

The CPT SEND service uses the acm_msend value, and the CPT RECEIVE service uses the acm_mrecv value. These values indicate the maximum number of user data bytes that can be transferred by the application in a single SEND or RECEIVE request to the transport provider. The user application is not limited to these values within the data transfer services. However, it is important to note that multiple transport provider or API requests are issued to complete the caller's request. Information on queue and buffer size is in the descriptions of **RECEIVE** on page 3-56 and **SEND** on page 3-65.

Initially, the tuning of data transfer storage may not be a concern. However, the ability to control storage allocation can prove beneficial to the application or CICS region. Queue size can increase data transfer throughput. You should consider enabling the statistics option to gather CPT statistical information. This information can set the SEND or RECEIVE queue and buffer size values.

The LISTEN service can modify the data transfer buffer allocation values. These values are negotiated with the transport provider and, depending on the site configuration, can be reduced. Any application dependent on these values should check them on return. These values are generally not modified when giving reasonable numbers. However, it is advisable to check with the site administrator for the maximum values of the API transport services.

A number of arguments are not set by the calling application, but are returned to the caller. These values represent information related to the client connection and can be used by the application. The local port, host name, and IP address are returned as well as the client's corresponding values. An ACM is passed the started transaction when an acm_trnid is specified in the caller's listen argument list.

200801-024040-200100

Client-Data Listener Option

The option ACMOPTN1_LTRAN is used in conjunction with acm_timeo and is mutually exclusive of the use of the acm_trnid field. ACMOPTN1_LTRAN indicates to the LISTEN service that the connecting client application will specify what server functions to execute. When the LISTEN service receives a CONNECT request and ACMOPTN1_LTRAN is specified, it uses a partial record timed RECEIVE (see RECEIVE service options) to get the client's data. It uses acm_timeo to know how long to wait for the client data which can be in any of these formats:

```
TRAN
TRAN,UUUUUUUUUUUU
TRAN,UUUUUUUUUUUU,IC,HMMSS
TDQN,UUUUUUUUUUUU,TD
TRAN,,IC,HHMMSS
TDQN,,TD
```

PARAMETER	DESCRIPTION
TRAN	A 1- to 4-character transaction ID to start, passing clnt_parm to the started transaction
UUUUUUUUUUUU	A 1- to 35-byte user data and is passed to the started transaction or written to the transient data queue in the field clnt_data.
IC	Specifies that TRAN is to be started in HHMMSS; if left blank, startup is immediate.
HHMMSS	Hours, minutes, and seconds for IC option.
TD	Indicates that clnt_parm will be written into the transient data queue, TDQN.

```
/*                                                          */
/*                C CLNT_PARM Definition                    */
/*                          */
typedef struct
    {
     int  *cp_token      ;  /* token                        */
    char  cp_fill1 [16] ;  /* Reserved              */
    char  cp_data  [36] ;  /* client data           */
    short cp_domain     ;  /* family                */
    short cp_rport      ;  /* remote port           */
    } client_parm;
```


Note: Using this option puts a listener at risk of being tied up until the client actually sends the data. Set acm_timeo with this fact in mind. The new trace flag, ACMTRAC2_CLTD, can optionally be specified to trace the CLNT_PARM written to TDQN transient data queue via CPT918I.

Example:

This example initiates a single-threaded server to listen on port 1234. The token is loaded from the ACM and used by all of the following CPT service requests. The `acm_rtncd` is checked to determine `LISTEN` service request completion status.

This sample program is generally not the preferred server model. The problem is that after returning from the `LISTEN` service the application blocks additional incoming connection requests. A better example of this facility is shown in the next example. This single-threaded server model is really only suitable for connections of a very short time duration.

```
#include  <t09ksacm.h>
#include  <t09ksrcs.h>

void main()
{
    struct acm_stru
        cpt_acm = {2, 0, 0, NULL, NULL, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        NULL,0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};

    /*
    *  CPT Connection Management Service request
    */

    cpt_acm.acm_lport = 1234;
    cpt_acm.acm_proto = ACMPROTO_TCP;
    t09flstn (&cpt_acm);
    if (cpt_acm.acm_rtncd != 0)
    {
        /* process CPT LISTEN service error and terminate transaction */
    }

    cpt_adt.adt_token = cpt_acm.acm_token;

    while (data)
    {
        /*
        * Application and CPT data transfer (SEND/RECEIVE) processing
        */
    }

    /*
    *  CPT Connection Release
    */

    /*
    *  Terminate Transaction
    */

    End:
    EXEC CICS RETURN;
}
```

200801-024040-200100

Example:

This example initiates a multi-threaded server to listen on port 1234. The token is loaded from the ACM and provided to a data processing transaction. The `acm_rtncd` is checked to determine `LISTEN` service request completion status. This server loops initiating client connections and starting data processing transactions.

This sample program differs from the previous example in that data transfer is not performed by the listening transaction, but by a different transaction. This allows for a more efficient server program. The server application is better able to respond quickly to new connection requests, because it is not involved in the tedious task of data transfer or connection management after the initialization connection.

This example does not utilize the optional `GIVE` facility management service. The `GIVE` service is beneficial if this task was expected to terminate before the data processing transaction initiated. However, since this task is not expected to terminate any abnormal termination would release all CPT resources currently associated with this task.

```
#include  <t09ksacm.h>
#include  <t09ksrcs.h>

void main()
{
    struct acm_stru
        cpt_acm = {2, 0, 0, NULL, NULL, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        NULL,0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};

    while (true)
    {
        /*
        *  CPT Connection Management Service request
        */

        cpt_acm.acm_lport = 1234;
        cpt_acm.acm_proto = ACMPROTO_TCP;
        t09flstn (&cpt_acm);
        if (cpt_acm.acm_rtncd != 0)
        {
            /* process CPT LISTEN service error and terminate transaction */
        }

        cpt_adt.adt_token = cpt_acm.acm_token;

        /*
        *  CPT Facility Management Give service request
        */

        EXEC CICS START TRANSID(trans-id) FROM(cpt_adt.adt_token);
    }

    /*
    *  CPT Connection Release
    */

    /*
    *  Terminate Transaction
    */

    End:
    EXEC CICS RETURN;
}
```

Example:

In this example a multi-threaded server listens for connections resolved to the transport service name DISCARD. An acm_trnid is specified that causes the LISTEN service to start a data processing transaction. Control is not returned to this program until some error has occurred. The acm_rtncd is checked to determine LISTEN service request completion status.

This sample program differs from the previous example in that a START command for a new transaction is performed by the LISTEN service and not by the user application. Also, control is not returned to the calling application until a failure occurs. Generally, this failure is due to termination of CICS, CPT, or the transport provider (API).

In this example, transaction SRV3 is automatically started by the LISTEN service for each connection established on the DISCARD port.

```
#include  <t09ksacm.h>
#include  <t09ksrcs.h>

void main()
{
    struct acm_stru
        cpt_acm = {2, 0, 0, NULL, NULL, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        NULL,0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};

    /*
    *  CPT Connection Management Service request
    */

    memcpy(cpt_acm.acm_srvce, "DISCARD", 7);
    memcpy(cpt_acm.acm_trnid, "SRV3", 4);
    t09flstn (&cpt_acm);
    if (cpt_acm.acm_rtncd != 0)
    {
        /* process CPT LISTEN service return code */
    }

    /*
    *  Terminate Transaction
    */

    End:
    EXEC CICS RETURN;
}
```

200801-024040-200100

RCVFROM

The RCVFROM service lets you develop connectionless client and server applications. This service is UDP only. The RCVFROM service provides these basic functions:

- Establishes a UDP server endpoint represented by a new token and starts receiving datagrams on a user-specified well-known port. Indicate this function to the RCVFROM service by passing an adt_token equal to zero. RCVFROM then creates all the internal control blocks and the RCVFROM buffer queue. Even though the SENDTO buffer queue is not allocated for this endpoint (token) until the SENDTO service is called, the SENDTO buffer size and number must be specified at this time because they are negotiated with the transport provider and recorded in the internal CPT control blocks at endpoint creation time. Upon return from the RCVFROM service, adt_token contains the token value to be passed to subsequent RCVFROM and SENDTO service calls.
- Receives a datagram at a previously established UDP endpoint represented by an existing token. This functionality makes the RCVFROM service call just a data transfer call that can be used by a client or server application. The RCVFROM buffer queue is only allocated upon the first call to the RCVFROM service, whether or not adt_token is equal to zero.

UDP tokens created with the RCVFROM or SENDTO services cannot be passed to the TCP-only services, CONNECT, LISTEN, SEND, and RECEIVE. All other CPT service calls are available to UDP applications.

The non-blocking option of the RCVFROM service (ADTOPCD1=ADTOPTN1_NBLKR), allows applications to be developed that can poll a well-known UDP port or send to a remote UDP server and then make a predetermined number of RCVFROM calls to get back a response. Given the general unreliable nature of UDP, not blocking on a RCVFROM call can build in some flexibility with regards to handling lost datagrams.

This table describes the arguments for the RCVFROM service:

INCLUDE STATEMENT	STRUCT NAME	SIZE	CREATED BY
t09ksadt.h	adt_stru	644 (X'284')	User application

20-024040-200100

Structure

This is the structure of the RCVFROM service in C language:

```
#ifndef __t09ksadt__
#define __t09ksadt__
 /* t09ksadt.h --  Declare CPT-dependent structures    */

 /*     ACCCSECT UPDTID=A1000000                        */

 /*                                                    */
 /*  Define Data Transfer (ADT) Structure               */
 /*                                                    */

typedef struct
     {
     short adt_vers     ;  /* adt block version number  */
     short adt_func     ;  /* Request function type     */
     int  *adt_token    ;  /* Data transfer token       */
     void *adt_buffa    ;  /* Data buffer address       */
     int   adt_buffl    ;  /* Buffer size/data length   */
      int   adt_rtncd    ;  /* Return code               */
      int   adt_dgncd    ;  /* Diagnostic code           */
      int   adt_stats    ;  /* Statistics options        */
      int   adt_trace    ;  /* Trace options             */
      int   adt_qsend    ;  /* Queue size - send         */
      int   adt_msend    ;  /* Buffer size - send        */
      int   adt_qrecv    ;  /* Queue size - recv         */
      int   adt_mrecv    ;  /* Buffer size - recv        */
      int   adt_timeo    ;  /* Timed receive - secs      */
      int   adt_rsvd1    ;  /* reserved (in use)         */
      int   adt_rsvd2    ;  /* reserved (in use)         */
      int   adt_rsvd3    ;  /* reserved (in use)         */
      int   adt_rsvd4    ;  /* reserved (in use)         */
      short adt_lport     ;  /* Local port               */
      short adt_rport     ;  /* Remote port              */
      short adt_rsvd5     ;  /* reserved (in use)        */
      char  adt_srvce [36];  /* Service name             */
      short adt_sepc     ;  /* # of sep characters       */
      char  adt_sep1     ;  /* 1st/only sep character    */
      char  adt_sep2     ;  /* 2nd sep character         */
      short adt_rsvd6     ;  /* reserved                 */
      int   adt_laddr    ;  /* Local host ip address     */
      int   adt_raddr    ;  /* Remote host ip address    */
      char  adt_lname[256];  /* Local host name          */
      char  adt_rname[256];  /* Remote host name         */
      int   adt_ucntx    ;  /* User context field        */
      char  adt_optn4    ;  /* Special options - unused  */
      char  adt_optn3    ;  /* Special options - unused  */
      char  adt_optn2    ;  /* Special options - byte 2  */
      char  adt_optn1    ;  /* Special options - byte 1  */
      } adt_stru;

      /*-------------------------------------------*/
       /*       CONTROL BLOCK VERSION NUMBER       */
      /*-------------------------------------------*/
#define ADT_VERSN       2
```

200801-024040-200100

```
    /*-------------------------------------------*/
     /*       RETURN CODE AT END-OF-FILE          */
    /*-------------------------------------------*/
#define NO_MORE_DATA   65  /* Remote Requests Orderly Release */

    /*-------------------------------------------*/
     /*            STATISTICS OPTIONS             */
    /*-------------------------------------------*/
#define ADTSTATS_CONN    0x01 /* CONNECTION STATISTICS */
#define ADTSTATS_TERM    0x02 /* TERMINATION STATISTICS*/

    /*-------------------------------------------*/
     /*            TRACE LOG REQUESTS             */
    /*-------------------------------------------*/
#define ADTTRAC1_NTRY   0x01    /*    ENTRY POINTS     */
#define ADTTRAC1_ARGS   0x02    /*     ARGUMENTS       */
#define ADTTRAC1_RECV   0x04    /*       TRECV         */
#define ADTTRAC1_SEND   0x08    /*       TSEND         */
#define ADTTRAC1_TERM   0x10    /*    TERMINATION      */
#define ADTTRAC1_PASS   0x20    /*     GIVE/TAKE       */
#define ADTTRAC1_CLSE   0x40    /*       CLOSE         */
#define ADTTRAC1_TERR   0x80    /*     TPLERRORS       */
#define ADTTRAC2_TOKN   0x01    /*       TOKENS        */
#define ADTTRAC2_TPL    0x02    /*         TPL         */
#define ADTTRAC2_RLSE   0x04    /*       RELEASE       */
#define ADTTRAC2_STOR   0x08    /*       STORAGE       */

    /*-------------------------------------------*/
     /*            SPECIAL OPTIONS                */
    /*-------------------------------------------*/
#define ADTOPTN1_TYPSP  0x01   /* record by separators */
#define ADTOPTN1_TYPLL  0x02   /* record by ll prefix  */
#define ADTOPTN1_BLCKS  0x04   /* blocking send        */
#define ADTOPTN1_TMPRT  0x08   /* timed partial receive*/
#define ADTOPTN1_TMRCV  0x10   /* timed full receive   */
#define ADTOPTN1_NBLKR  0x40   /* non-blocking receive */
#define ADTOPTN1_DODNR  0x80   /* do DNR calls for UDP */
#define ADTOPTN2_NOSTP  0x40   /* no stripping ll/sep  */
#define ADTOPTN2_VLIST  0x80    /* vector list         */

    /*-------------------------------------------*/
     /*       CPT DATA TRANSFER SERVICE           */
    /*-------------------------------------------*/
#if defined (IBMC)
 #pragma linkage(t09frecv,OS)
 #pragma linkage(t09fsend,OS)
#else
 #define t09frecv __asm_t09frec
 #define t09fsend __asm_t09fsen
#endif

void t09frecv (), t09fsend ();

#endif
```

PARAMETER	DESCRIPTION
adt_vers	Version Indicates the CPT version number of the argument list used by the calling program. This required field must be set to ADT_VERSN for this release of CPT. Default: None
adt_func	Function code Indicates the function or callable service ID requested by the application program this field should not be set by the application, but rather is initialized by the TRUE interface stub. Default: None
adt_token	Data transfer token Specifies a token that represents a UDP endpoint. If the ADT is being passed in a call to either the RCVFROM or SENDTO service, then the token can be zero, indicating to either service, to first create a token before sending or receiving a datagram. If the token is not zero, then it must be a token created previously by either the RCVFROM or SENDTO service. It is not necessary or efficient to create a token every time a CICS transaction calls the UDP data transfer services. It is an error to pass a TCP token to the UDP data transfer service routines, RCVFROM and SENDTO. Conversely it is an error to pass a UDP token to the TCP data transfer routines, RECEIVE and SEND.
adt_buffa	User data address Indicates the storage address into which the UDP datagram is received (RCVFROM service). This is a contiguous segment of storage accessible to the user task. The content of all user data is application dependent, and not interpreted by either CPT nor the transport provider. The storage area can be aligned on any boundary convenient for the application program. Default: 0
adt_buffl	User data length Indicates the length (in bytes) of the buffer specified in ADTBUFFA which is to receive (RCVFROM service) the UDP datagram. If the incoming datagram does not fit into adt_buffa for a length of adt_buffl, then the warning, CPTWNEOM will be passed back to the caller in adt_rtncd, indicating that more RCVFROM calls will be required to get the entire datagram. adt_buffl will indicate the actual length returned in adt_buffl. It is an error to call the RCVFROM service with an adt_buffl of zero.
adt_rtncd	Return code Indicates the return code set by the RCVFROM service. Default: 0
adt_dgncd	Diagnostic code Indicates the diagnostics code set by the RCVFROM service. This value generally indicates a transport provider return code. Default: 0

200801-024040-200100

PARAMETER	DESCRIPTION
adt_stats	Specifies logging options for the application program. The facility can be used for debugging and tuning during development. Note that this field has no meaning for TCP connections, since the statistics are set at connection establishment time via the ACMSTATS field. ◆ acmstats_conn – Specifies that messages be generated on the closing of a UDP token. These messages are generated by the CPT CLOSE service. The message numbers are CPT802I and CPT803I. ◆ acmsstats_term – Specifies that messages be generated on terminating an established connection. These messages are generated by the CPT CLOSE service. The message numbers are CPT807I, CPT808I and CPT809I. Default: 0 (No statistics logging)
adt_trac1	Specifies trace logging options for the application program. The facility can be used for debugging during development. ◆ ADTTRAC1_NTRY – Specifies that a message be generated on entry to a CPT service routine. The message is generated by all CPT service routines. The message number associated with this option is CPT901I. ◆ ADTTRAC1_ARGS – Specifies that a hex dump of the caller's arguments be generated on entry and exit of an CPT service routine. Messages are generated by the corresponding service routine. Message numbers associated with this option are CPT902I, CPT903I, CPT904I, CPT911I, CPT912I, CPT921I,CPT922I, CPT924I, CPT925I, CPT930I, and CPT932I. ◆ ADTTRAC1_RECV – Specifies that a hex dump of the transport provider's input (RECEIVE) data be logged. Messages are generated by the CPT RECV service. The message number associated with this option is CPT913I. ◆ ADTTRAC1_SEND – Specifies that a hex dump of the transport provider's output (SEND) data be logged. Messages are generated by the CPT SEND service. The message number associated with this option is CPT914I. ◆ ADTTRAC1_TERM – Specifies that a message be generated on termination of an CPT service routine. Messages are generated by all CPT service routines. The message number associated with this option is CPT908I. ◆ ADTTRAC1_PASS – Specifies that a hex dump of resources related to a passed connection be logged. Messages are generated by the CPT GIVE and TAKE services. The message numbers associated with this option are CPT919I and CPT920I. ◆ ADTTRAC1_CLSE – Specifies that a hex dump of resources related to a CLOSE processing of a connection be logged. Message are generated by the CPT CLOSE service. The message number associated with this option is CPT906I. ◆ ADTTRAC1_TERR – Specifies that a hex dump of a transport provider API parameter list that fails successful completion be logged. The message number associated with this option is CPT400I.

20 024040-200100

PARAMETER	DESCRIPTION
adt_trac2	◆ ADTTRAC2_TOKN – Specifies that a hex dump of the connection token be logged. Messages are generated by all services routines on entry. The message number associated with this option is CPT909I. ◆ ADTTRAC2_TPL – Specifies that a hex dump of a transport provider API parameter list be logged. The message numbers associated with this option are CPT917I and CPT935I. ◆ ADTTRAC2_RLSE – Specifies that a message indicating a transport provider release indication be logged. Messages are generated by various CPT service routines. The message number associated with this option is CPT414I. ◆ ADTTRAC2_STOR – Specifies that a hex dump of storage management arguments be logged. Messages are generated by various CPT service routines. The message number associated with this option is CPT928I and CPT929I. Default: 0 (no trace logging)
adt_qsend	API send queue size (used when adt_token=0) Specifies the maximum number of uncompleted RCVFROM requests that can be queued by the application to the transport provider (API). This value lets applications control output processing and can affect throughput rates. The value is negotiated with the transport provider and can be modified by the transport provider. The total allocation for output processing is the product of the RCVFROM queue and buffer size values and cannot exceed 61K. Default: 4
adt_msend	API send buffer size (used when adt_token=0) Specifies the maximum number of user data bytes that can be transferred by the application in a single RCVFROM request to the transport provider (API). This value lets applications control output processing and can affect throughput rates. The value is negotiated with the transport provider and can be modified by the transport provider. The total allocation for output processing is the product of the RCVFROM queue and buffer size values and cannot exceed 61K. Default: 4096
adt_qrecv	API RECEIVE queue size (used when adt_token=0 Specifies maximum number of uncompleted RCVFROM requests that can be queued by the application to the transport provider (API). This value lets applications control input processing and can affect throughput rates. The value is negotiated with the transport provider and can be modified by the transport provider. The total allocation for input processing is the product of the RCVFROM queue and buffer size values and cannot exceed 61K. Default: 4
adt_mrecv	API RECEIVE buffer size (used when adt_token=0) Specifies the maximum number of user data bytes that can be transferred by the application in a single RCVFROM request to the transport provider (API). This value lets applications control input processing and can affect throughput rates. The value is negotiated with the transport provider and can be modified by the transport provider. The total allocation for input processing is the product of the RCVFROM queue and buffer size values and cannot exceed 61K. Default: 4096

200801-024040-200100

PARAMETER	DESCRIPTION
adt_timeo	RECEIVE time out value Not used by the RCVFROM service Default: 0
adt_lport	Local well-known service port (used when adt_token=0) Indicates the local transport layer port that the calling application will be receiving (RCVFROM) datagrams on. If the SENDTO service creates the token, this port number is assigned by the transport layer and returned to the caller. if the RCVFROM service creates the token, this is the well-known port requested by the caller. If the RCVFROM service is creating the token, this value or the transport layer service name (adt_srvce) must be specified. This field is an unsigned positive integer with a maximum value of 65,534. The value must be unique for each server application. Default: None
adt_rport	Remote port Indicates the remote transport layer port on which the incoming datagram originated. This value is returned to the caller of the RCVFROM service and may be different for each datagram received. This field is an unsigned positive integer with a maximum value of 65,534. Default: None
adt_srvce	Transport layer service name (used when adt_token=0) Indicates the local transport layer service name. This string is mapped into a transport layer address or port, through internal DNR. The resolved value is the well-known port to which the client application sends (SENDTO) UDP datagrams. This field has a maximum length of 36 bytes. The value must be unique for each server application. This field is optional and is not modified by the RCVFROM service. Default: None
adt_sepc	Number of separator characters for option ADTOPTN1_TYPSP. Not used in the RCVFROM service. Default: None
adt_sep1	First or only separator character for option ADTOPTN1_TYPSP. Not used in the RCVFROM service. Default: None
adt_sep2	Second character or separator sequence for option ADTOPTN1_TYPSP. Not used in the RCVFROM service. Default: None
adt_laddr	Local host IP address Indicates the local host internet address. This field is an unsigned four-byte integer value. The local host internet address is returned to the caller of the RCVFROM Default: None
adt_raddr	Remote host IP address Indicates the remote host internet address of the sender of the incoming UDP datagram. This value is returned to the caller of the RCVFROM service and may be different for each datagram received. This field is an unsigned four-byte integer value. Default: None

PARAMETER	DESCRIPTION
adt_lname	Local host name Indicates the local host internet name. This field is a 255-byte character string that is padded with blanks. The local host internet name is returned to the caller of the RCVFROM service.
adt_rname	Remote host name Indicates the remote host internet name. This field is a 255-byte character string that is padded with blanks. It is only resolved through internal DNR calls and returned to the caller of the UDP data transfer service routines (RCVFROM and SENDTO) if the adt_optn1 flag, ADTOPTN1_DODNR is specified. This is to prevent the DNR call overhead on every UDP data transfer call. Default: None
adt_ucntx	One word of user context Specifies one arbitrary word of user context to be associated with the endpoint. The information provided is not interpreted by CPT, and is merely saved with other endpoint information. Default: 0 (no user context)
adt_optn2	Special options - byte 2
adt_optn1	Specifies data transfer options These are the ADT options that apply to UDP data transfer requests: ◆ ADTOPTN1_DODNR– Execute internal DNR calls during UDP data transfer service routine calls (RCVFROM and SENDTO) to resolve remote IP addresses into IP names in the ADTRNAME field. ◆ ADTOPTN1_NBLKR – Do not block on a call to the RCVFROM service. If no datagrams are currently available, the return code, CPTWBLCK, will be returned in ADTRTNCD. Default: None **Note:** These options can be toggled on every UDP data transfer call even if the caller is using the same token.

200801-024040-200100

Network Considerations

The ADT is a common data structure used for both client and server UDP applications. There are common and unique values specified for a particular service request.

The table describes network considerations for C API:

NAME	SERVER CONDITIONS FOR RCVFROM	CLIENT CONTITIONS FOR SENDTO
adt_lport	Local server well-known port selected by user application.	Local assigned transport provider port returned to user application.
adt_rport	Remote client transport provider port returned to user application.	Remote server transport provider well-known port selected by user application.
adt_srvce	Local server transport provider service name selected by user application.	Remote server transport provider service name selected by user application.

NAME	SERVER CONDITIONS FOR RCVFROM	CLIENT CONTITIONS FOR SENDTO
adt_raddr	Remote IP host address returned to user application.	Remote IP host address selected by or returned to user application. The client must specify this field or adt_rname.
adt_lname	Local IP host name returned to user application.	Local IP host name Returned to user application.
adt_rname	Remote IP host name returned to user application only if ADTOPTN1_DODNR is specified in adt_optn1.	Remote IP host name selected by or returned to the user application. The client must specify this field or adt_raddr. If adt_raddr is used adt_rname will only be returned if ADTOPTN1_DODNR is specified in adt_optn1.

Return Codes

The RCVFROM service return and diagnostic codes indicate the result of the execution. These values are in the adt_rtncd and adt_dgncd within the ADT. The diagnostic code is optional and generally indicates a transport provider return code. Read **Appendix C – MESSAGES AND CODES** in the ***CICS Programmer's Toolkit Installation and Administration Guide*** for the return code cross reference table.

This table describes the RCVFROM service return codes:

RETURN	DIAGNOSTIC CODE	DESCRIPTION
CPTIRCOK		Successful.
CPTEVERS		Control block version number is not supported.
CPTETOKN		Specified token is not valid.
CPTEBUFF		Buffer address and/or length is invalid.
CPTENAPI	Yes	Transport provider API is not available.
CPTETERM	Yes	Environment is being terminated.
CPTERLSE	Yes	Release indication.
CPTEDISC	Yes	Disconnect indication.
CPTEINTG	Yes	Transport provider API integrity error.
CPTEENVR	Yes	Transport provider API environment error.
CPTEFRMT	Yes	Transport provider API format error.
CPTEPROC	Yes	Transport provider API procedure error.
CPTABEND		Abnormal exception occurred.
CPTEOTHR	Yes	An undefined exception occurred.

024040-200100

Example:

This example receives data from a remote host. The the `adt_rtncd` is checked to determine `RCVFROM` service request completion status.

```
#include  <t09ksadt.h>
#include  <t09ksrcs.h>

void main()
{
    struct    adt_stru
              cpt_adt = {ADT_VERSN, 0, 0, NULL, NULL, 0, 0, 0, 0};

    char      msgarea[256];

    /*
    *  Identify Service
    */

    cpt_adt.adt_lport = 1680;

    /*
    *  Data Processing Routine
    */

    while (data)
    {
        cpt_adt.adt_buffa = &msgarea;
        cpt_adt.adt_buffl = sizeof (msgarea);
        t09frcfr (&cpt_adt);
        if (cpt_adt.adt_rtncd != 0)
        {
            /*
            *  process CPT RECEIVE service error
            */
        }
    }
    /*
    *  CPT Terminate Endpoint
    */
    /*
    *  Terminate Transaction
    */

    End:
    EXEC CICS RETURN;
```

200801-024040-200100

RECEIVE

The RECEIVE service receives data from a peer transport user connected to an endpoint. The RECEIVE service receives data as input on a connection-mode (TCP) endpoint only.

To invoke the RECEIVE service, a user application is required to first build an Argument for Data Transfer (ADT) and then to issue a call to the RECEIVE routine. The ADT contains the version number, connection token, user buffer address, and length. When the RECEIVE service completes, the buffer length field is updated to reflect the amount of data processed by the RECEIVE service.

This table describes the arguments for the RECEIVE service:

INCLUDE STATEMENT	STRUCT NAME	SIZE	CREATED BY
t09ksadt.h	adt_stru	644 (X'284')	User application

Structure

This is the structure of the RECEIVE service in C language:

```
#ifndef __t09ksadt__
#define __t09ksadt__
 /* t09ksadt.h --  Declare CPT-dependent structures    */

 /*     ACCCSECT UPDTID=A1000000                        */

 /*                                                    */
 /*  Define Data Transfer (ADT) Structure               */
 /*                                                    */

typedef struct
     {
    short adt_vers     ;  /* adt block version number  */
    short adt_func     ;  /* Request function type     */
    int  *adt_token    ;  /* Data transfer token       */
    void *adt_buffa    ;  /* Data buffer address       */
    int   adt_buffl    ;  /* Buffer size/data length   */
     int   adt_rtncd    ;  /* Return code               */
     int   adt_dgncd    ;  /* Diagnostic code           */
     int   adt_stats    ;  /* Statistics options        */
     int   adt_trace    ;  /* Trace options             */
     int   adt_qsend    ;  /* Queue size - send         */
     int   adt_msend    ;  /* Buffer size - send        */
     int   adt_qrecv    ;  /* Queue size - recv         */
     int   adt_mrecv    ;  /* Buffer size - recv        */
     int   adt_timeo    ;  /* Timed receive - secs      */
     int   adt_rsvd1    ;  /* reserved (in use)         */
     int   adt_rsvd2    ;  /* reserved (in use)         */
     int   adt_rsvd3    ;  /* reserved (in use)         */
     int   adt_rsvd4    ;  /* reserved (in use)         */
     short adt_lport    ;  /* Local port                */
     short adt_rport    ;  /* Remote port               */
     short adt_rsvd5    ;  /* reserved (in use)         */
```

```
    char  adt_srvce [36];  /* Service name              */
    short adt_sepc     ;  /* # of sep characters       */
    char  adt_sep1     ;  /* 1st/only sep character    */
    char  adt_sep2     ;  /* 2nd sep character         */
    short adt_rsvd6    ;  /* reserved                  */
    int   adt_laddr    ;  /* Local host ip address     */
    int   adt_raddr    ;  /* Remote host ip address    */
    char  adt_lname[256];  /* Local host name           */
    char  adt_rname[256];  /* Remote host name          */
    int   adt_ucntx    ;  /* User context field        */
    char  adt_optn4    ;  /* Special options - unused  */
    char  adt_optn3    ;  /* Special options - unused  */
    char  adt_optn2    ;  /* Special options - byte 2  */
    char  adt_optn1    ;  /* Special options - byte 1  */
    } adt_stru;

    /*-----------------------------------------------*/
     /*       CONTROL BLOCK VERSION NUMBER          */
    /*-----------------------------------------------*/
#define ADT_VERSN       2

    /*-----------------------------------------------*/
     /*       RETURN CODE AT END-OF-FILE            */
    /*-----------------------------------------------*/
#define NO_MORE_DATA   65  /* Remote Requests Orderly Release */

    /*-----------------------------------------------*/
     /*            STATISTICS OPTIONS               */
    /*-----------------------------------------------*/
#define ADTSTATS_CONN    0x01 /* CONNECTION STATISTICS */
#define ADTSTATS_TERM    0x02 /* TERMINATION STATISTICS*/

    /*-----------------------------------------------*/
     /*            TRACE LOG REQUESTS               */
    /*-----------------------------------------------*/
#define ADTTRAC1_NTRY  0x01   /*    ENTRY POINTS     */
#define ADTTRAC1_ARGS  0x02   /*     ARGUMENTS       */
#define ADTTRAC1_RECV  0x04   /*       TRECV         */
#define ADTTRAC1_SEND  0x08   /*       TSEND         */
#define ADTTRAC1_TERM  0x10   /*    TERMINATION      */
#define ADTTRAC1_PASS  0x20   /*     GIVE/TAKE       */
#define ADTTRAC1_CLSE  0x40   /*       CLOSE         */
#define ADTTRAC1_TERR  0x80   /*     TPLERRORS       */
#define ADTTRAC2_TOKN  0x01   /*       TOKENS        */
#define ADTTRAC2_TPL   0x02   /*         TPL         */
#define ADTTRAC2_RLSE  0x04   /*       RELEASE       */
#define ADTTRAC2_STOR  0x08   /*       STORAGE       */

    /*-----------------------------------------------*/
     /*            SPECIAL OPTIONS                  */
    /*-----------------------------------------------*/
#define ADTOPTN1_TYPSP 0x01   /* record by separators */
#define ADTOPTN1_TYPLL 0x02   /* record by ll prefix  */
#define ADTOPTN1_BLCKS 0x04    /* blocking send       */
#define ADTOPTN1_TMPRT 0x08   /* timed partial receive*/
#define ADTOPTN1_TMRCV 0x10   /* timed full receive   */
#define ADTOPTN1_NBLKR 0x40   /* non-blocking receive */
#define ADTOPTN1_DODNR 0x80   /* do DNR calls for UDP */
#define ADTOPTN2_NOSTP 0x40   /* no stripping ll/sep  */
```

200801-024040-200100

```
#define ADTOPTN2_VLIST  0x80    /* vector list          */

    /*-------------------------------------------*/
     /*       CPT DATA TRANSFER SERVICE          */
    /*-------------------------------------------*/
#if defined (IBMC)
 #pragma linkage(t09frecv,OS)
 #pragma linkage(t09fsend,OS)
#else
 #define t09frecv __asm_t09frec
 #define t09fsend __asm_t09fsen
#endif

 void t09frecv (), t09fsend ();

#endif
```

PARAMETER	DESCRIPTION
adt_vers	Version Indicates the CPT version number of the argument list used by the calling program. This required field must be set to ADT_VERSN for this release of CPT. Default: None
adt_func	Function code Indicates the function or callable service ID requested by the application program. This field is not set by the application, but is initialized by the TRUE interface stub program. Default: None
adt_token	Data transfer token Specifies a token that represents the connection. A token is obtained from a connection management request. The token is required. Default: None
adt_buffa	User data address Indicates the storage address into which network data is placed. This is a contiguous segment of storage, accessible to the user task. The content of all user data is application-dependent, and is not interpreted by either CPT or by the transport provider. The storage area can be aligned on any boundary convenient for the application program. Default: 0
adt_buffl	User data length Indicates the length (in bytes) of user data in the storage area as identified by the adt_buffa operand. The length is updated when the request is completed to reflect the actual length of user data received. This field must be interpreted on completion to determine the amount of data actually received. If a RECEIVE request is issued with a zero length, an error is detected and the request fails. Default: 0
adt_rtncd	Return code Indicates the return code set by the RECEIVE service. This value indicates the success or failure of the service. Default: 0
adt_dgncd	Diagnostic code Indicates the diagnostic code received by the RECEIVE service request. This value generally indicates a transport provider return code. Default: 0

PARAMETER	DESCRIPTION
adt_stats	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
adt_trace	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
adt_qsend	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
adt_msend	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
adt_qrecv	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
adt_mrecv	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
adt_timeo	Select wait seconds Must be specified with these options: ◆ ADTOPTN1_TYPLL ◆ ADTOPTN1_TYPSP ◆ ADTOPTN1_TMRCV ◆ ADTOPTN1_TMPRT Specifying any of the above options on a RECEIVE call with an adt_timeo=ZERO will result in CPTETIME being returned in adt_rtncd. Default: None
adt_lport	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
adt_rport	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
adt_srvce	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
adt_sepc	Number of sep characters If adt_sepc is not equal to one or two, CPTESEP# will be returned in adt_rtncd. Default: None
adt_sep1	First or only sep character Default: None
adt_sep2	Second sep character. Default: None
adt_laddr	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
adt_raddr	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
adt_lname	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
adt_rname	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.

200801-024040-200100

PARAMETER	DESCRIPTION
adt_ucntx	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
adt_optn2	◆ ADTOPTN1_NOSTP – Do not strip record delimiter sequence This can be used with ADTOPTN1_TYPSP or ADTOPTN1_TYPLL to return the actual separator sequence or LL field in the buffer pointed to by adt_buffa. ◆ ADTOPTN2_VLIST – Currently internal use only
adt_optn1	This field specifies data transfer options. These are the ADT options that apply to TCP data transfer requests: ◆ ADTOPTN1_NBLKR – Do not block on a call to the RECEIVE service If no data is currently available on the connection, CPTWBLCK will be returned in adt_rtncd. ◆ ADTOPTN1_TMRCV – Timed full record RECEIVE These fields (along with other required ADT fields) are used to request a timed full record RECEIVE: ADTOPCD1 = ADTOPTN1_TMRCV adt_timeo > zero adt_buffl set to the length expected If the time limit expires before receiving any or all of the data specified by adt_buffl, CPTWTIMO will be returned in adt_rtncd along with any data that was received. ◆ ADTOPTN1_TMPRT – Timed partial record RECEIVE These fields (along with other required ADT fields) are used to request a timed partial record RECEIVE: ADTOPCD1 = ADTOPTN1_TMPRT adt_timeo > zero adt_buffl set to maximum length expected If the time limit expires before receiving data, CPTWTIMO is returned in adt_rtncd. If the time limit expires and any data is received, the data, along with a zero adt_rtncd, will be returned to the caller. ◆ ADTOPTN1_TYPSP – SEP type RECEIVE These fields (along with other required ADT fields) are used to request a SEP type RECEIVE call: ADTOPCD1 = ADTOPTN1_TYPSP adt_sepc = 1 or 2 adt_sep1 = character adt_sep2 = character if adt_sepc = 2 adt_timeo> zero If the time limit expires and data is received, but no SEP characters are found, the data, along with an adt_rtncd of CPTWNSEP will be returned to the caller. ◆ ADTOPTN1_TYPLL – LL type RECEIVE These fields (along with other required ADT fields) are used to request a SEP type RECEIVE call: ADTOPCD1 = ADTOPTN1_TYPLL adt_timeo > zero If the time limit expires before receiving any or all of the data specified by the LL (first two bytes of the data stream), CPTWTIMO will be returned in adt_rtncd, along with any data that was received. ◆ ADTOPTN1_BLCKS – Block on SEND service call (not used by RECEIVE service). **Note:** It is an error to combine any of these RECEIVE service options: ADTOPTN1_NBLKR, ADTOPTN1_TYPLL, ADTOPTN1_TYPSP, ADTOPTN1_TMRCV, ADTOPTN1_TMPRT. An invalid combination causes CPTEOPTN to return in adt_rtncd. Default: None 1

Completion Information

The RECEIVE service completes normally when the data is moved from the transport provider buffer to the application program's storage area. A length is returned to the application program, which is set to the amount of data actually processed.

Normal completion of the RECEIVE service implies that data has been moved to the user buffer. This does not necessarily indicate the application request was completely satisfied, but that some amount of data was processed. The user application is required to load the adt_buffl field to determine the actual data received. The RECEIVE service returns control to the calling application on receipt of a full buffer, a partial buffer, or an error indication. Control is returned to the user application with a partial buffer to avoid a WAIT command within the RECEIVE service. Additional requests to the RECEIVE service may be required to completely satisfy the user application's requirement.

The presence of exceptions or error conditions do not always indicate serious errors. A user application should check the return code to determine proper flow control. The release indication return code is an example of a condition that is not necessarily a serious error. This exception specifies that the remote host has closed its half of the full-duplex data connection and will not send any additional data. This return code is acceptable, and generally indicates that graceful termination of the connection should begin.

On normal return to the application program, the general return code in adt_rtncd is set to zero (CPTIRCOK). The diagnostic code in adt_dgncd is always zero (CPTIRCOK). The length field (adt_buffl) indicates the amount of data processed.

If the RECEIVE service completes abnormally, some or no user data may have been sent to the peer transport user. The general return code in adt_rtncd, and the diagnostic code in adt_dgncd, indicate the nature of the failure. The diagnostic code (adt_dgncd) generally contains a specific code that is generated by the transport provider.

200801-024040-200100

Return Codes

The RECEIVE service return and diagnostic codes indicate the result of the execution. These values are in the adt_rtncd and adt_dgncd within the ADT. The diagnostic code is optional and generally indicates a transport provider return code. Read **Appendix C – MESSAGES AND CODES** in the ***CICS Programmer's Toolkit Installation and Administration Guide*** for the return code cross reference table.

This table describes the RECEIVE service return codes:

RETURN	DIAGNOSTIC CODE	DESCRIPTION
CPTIRCOK		Successful.
CPTEVERS		Control block version number is not supported.
CPTETOKN		Specified token is not valid.
CPTEBUFF		Buffer address and/or length is invalid.
CPTENAPI	Yes	Transport provider API is not available.
CPTETERM	Yes	Environment is being terminated.
CPTERLSE	Yes	Release indication.
CPTEDISC	Yes	Disconnect indication.
CPTEINTG	Yes	Transport provider API integrity error.
CPTEENVR	Yes	Transport provider API environment error.
CPTEFRMT	Yes	Transport provider API format error.
CPTEPROC	Yes	Transport provider API procedure error.
CPTABEND		Abnormal exception occurred.
CPTEOTHR	Yes	An undefined exception occurred.

Usage Information

The RECEIVE service receives normal data as input through a CPT connection. The is part of a byte stream being received over a connection (TCP).

Data is moved from the transport provider's storage area to the user application's storage area. Stream data might not be received with the same logical boundaries with which it was sent. However, the data arrives in the precise order in which it was sent. Possible fragmentation is a characteristic of stream data.

User applications may be required to issue multiple RECEIVE service requests to obtain all of the desired data. The data may arrive in particle segments. An application should be designed to handle such a situation. Additionally, users who write applications to process multiple record oriented data should consider including a mechanism to delimit the data. Design options can include

024040-200100

a logical length field at the beginning of a record, or a special field, or fields, at the end. This lets the application determine record boundaries.

The RECEIVE service request is a synchronous operation, which may require the application to be blocked. The RECEIVE queue (acm_qrecv) and buffer (acm_mrecv) sizes, as specified during connection initialization, control input queuing and buffering. The queue size represents the number of uncompleted RECEIVE requests that can be pending to the transport provider. The default value is generally sufficient to support most applications. The transport provider normally completes a RECEIVE request on transferring data into the user buffer. However, data over the network may be fragmented and may require multiple requests to retrieve all of the data. The RECEIVE service issues a WAIT command if no data is available.

The queue and buffer size values are specified during connection initialization and can be modified by either the LISTEN or CONNECT services. An application that is dependent on these values should validate the requested values, compared with those values returned within the ACM. The values are modified if the transport provider site administrator has configured limits and the application request exceeds those values. If the requested values are modified, the user should verify site definition statements for API transport services.

The version number (adt_vers) indicates the CPT release level in which this user application program is written. This required field must be set to ADT_VERSN and is validated by the RECEIVE service before processing the request.

The function code (adt_func) indicates the CPT callable service ID. The field is not initialized by a user application program and has little value to the application except for dump analysis. The function code identifies and maps an argument list with the error or trace log and dump analysis.

The token (adt_token) indicates the connection that is to receive data. This is a required field and is validated by the RECEIVE service before processing the request.

The data buffer address field adt_buffa is a full word. The application program is responsible for assuring that the residency mode of data areas it manages (for example, argument lists) is compatible with the addressing mode. The transport provider performs consistency checks on the addressing mode whenever a service request is issued. However, unpredictable results can occur before the transport provider performs this check.

The data buffer length is indicated by the adt_buffl field. This is a full word positive integer. The data buffer length field should be less than or equal to the maximum receive buffer values. However, if the data buffer length is greater than the maximum receive buffer, the RECEIVE service attempts to satisfy the user's request with multiple transport provider requests. On return from the RECEIVE service, the adt_buffl is updated with a value that indicates the number of bytes processed.

The adt_optns field specifies RECEIVE processing control options and provides a mechanism for event notification on return to the application program.

200801-024040-200100

Example:

This example establishes a connection and receives data from the remote host. The token is loaded from the ACM and used by all of the following CPT service requests. The the `acl_rtncd` is checked to determine `RECEIVE` service request completion status.

```
#include  <t09ksadt.h>
#include  <t09ksacm.h>
#include  <t09ksrcs.h>

void main()
{
    struct    adt_stru
              cpt_adt = {2, 0, 0, NULL, NULL, 0, 0, 0, 0};

    char      message[] = "Sample CPT C program application";

    /*
    *  CPT Connection Management service request
    */

    cpt_adt.adt_token = cpt_acm.acm_token;

    /*
    *  Data Processing Routine
    */

    while (data)
    {
        cpt_adt.adt_buffa = &msgarea;
        cpt_adt.adt_buffl = msglen;
        t09frecv (&cpt_adt);
        if (cpt_adt.adt_rtncd != 0)
        {
            /*
            *  process CPT RECEIVE service error
            */
        }
    }
    /*
    *  CPT Release Connection
    */
    /*
    *  Terminate Transaction
    */

    End:
    EXEC CICS RETURN;
```


024040-200100

SEND

The SEND service sends data to a peer transport user, connected to an endpoint. The SEND service sends data as output on a connection-mode (TCP) endpoint only.

To invoke the SEND service, a user application is required to first build an ADT and then to issue a call to the SEND routine. The ADT is required to contain the version number, connection token, user buffer address, and length. On completion of the SEND service, the buffer length field is updated to reflect the amount of data processed.

This table describes the arguments for the SEND service:

INCLUDE STATEMENT	STRUCT NAME	SIZE	CREATED BY
t09ksadt.h	adt_stru	28 (X'1C')	User application

Structure

This is the structure of the SEND service in C language:

```
#ifndef __t09ksadt__
#define __t09ksadt__
 /* t09ksadt.h --  Declare CPT-dependent structures     */

 /*     ACCCSECT UPDTID=A1000000                         */

 /*                                                     */
 /*  Define Data Transfer (ADT) Structure               */
 /*                                                     */

typedef struct
     {
    short adt_vers     ;  /* adt block version number  */
    short adt_func     ;  /* Request function type     */
    int  *adt_token    ;  /* Data transfer token       */
    void *adt_buffa    ;  /* Data buffer address       */
    int   adt_buffl    ;  /* Buffer size/data length   */
     int   adt_rtncd    ;  /* Return code               */
     int   adt_dgncd    ;  /* Diagnostic code           */
     int   adt_stats    ;  /* Statistics options        */
     int   adt_trace    ;  /* Trace options             */
     int   adt_qsend    ;  /* Queue size - send         */
     int   adt_msend    ;  /* Buffer size - send        */
     int   adt_qrecv    ;  /* Queue size - recv         */
     int   adt_mrecv    ;  /* Buffer size - recv        */
     int   adt_timeo    ;  /* Timed receive - secs      */
     int   adt_rsvd1    ;  /* reserved (in use)         */
     int   adt_rsvd2    ;  /* reserved (in use)         */
     int   adt_rsvd3    ;  /* reserved (in use)         */
     int   adt_rsvd4    ;  /* reserved (in use)         */
     short adt_lport    ;  /* Local port                */
     short adt_rport    ;  /* Remote port               */
     short adt_rsvd5    ;  /* reserved (in use)         */
```

200801-024040-200100

```
    char  adt_srvce [36];  /* Service name              */
    short adt_sepc      ;  /* # of sep characters       */
    char  adt_sep1      ;  /* 1st/only sep character    */
    char  adt_sep2      ;  /* 2nd sep character         */
    short adt_rsvd6     ;  /* reserved                  */
    int   adt_laddr     ;  /* Local host ip address     */
    int   adt_raddr     ;  /* Remote host ip address    */
    char  adt_lname[256];  /* Local host name           */
    char  adt_rname[256];  /* Remote host name          */
    int   adt_ucntx     ;  /* User context field        */
    char  adt_optn4     ;  /* Special options - unused  */
    char  adt_optn3     ;  /* Special options - unused  */
    char  adt_optn2     ;  /* Special options - byte 2  */
    char  adt_optn1     ;  /* Special options - byte 1  */
    } adt_stru;

    /*-------------------------------------------*/
     /*       CONTROL BLOCK VERSION NUMBER       */
    /*-------------------------------------------*/
#define ADT_VERSN       2

    /*-------------------------------------------*/
     /*       RETURN CODE AT END-OF-FILE         */
    /*-------------------------------------------*/
#define NO_MORE_DATA   65  /* Remote Requests Orderly Release */

    /*-------------------------------------------*/
     /*            STATISTICS OPTIONS            */
    /*-------------------------------------------*/
#define ADTSTATS_CONN    0x01 /* CONNECTION STATISTICS */
#define ADTSTATS_TERM    0x02 /* TERMINATION STATISTICS*/

    /*-------------------------------------------*/
     /*            TRACE LOG REQUESTS            */
    /*-------------------------------------------*/
#define ADTTRAC1_NTRY   0x01    /*    ENTRY POINTS    */
#define ADTTRAC1_ARGS   0x02    /*     ARGUMENTS      */
#define ADTTRAC1_RECV   0x04    /*       TRECV        */
#define ADTTRAC1_SEND   0x08    /*       TSEND        */
#define ADTTRAC1_TERM   0x10    /*    TERMINATION     */
#define ADTTRAC1_PASS   0x20    /*     GIVE/TAKE      */
#define ADTTRAC1_CLSE   0x40    /*       CLOSE        */
#define ADTTRAC1_TERR   0x80    /*     TPLERRORS      */
#define ADTTRAC2_TOKN   0x01    /*       TOKENS       */
#define ADTTRAC2_TPL    0x02    /*         TPL        */
#define ADTTRAC2_RLSE   0x04    /*      RELEASE       */
#define ADTTRAC2_STOR   0x08    /*      STORAGE       */

    /*-------------------------------------------*/
     /*            SPECIAL OPTIONS               */
    /*-------------------------------------------*/
#define ADTOPTN1_TYPSP  0x01   /* record by separators */
#define ADTOPTN1_TYPLL  0x02   /* record by ll prefix  */
#define ADTOPTN1_BLCKS  0x04    /* blocking send       */
#define ADTOPTN1_TMPRT  0x08   /* timed partial receive*/
#define ADTOPTN1_TMRCV  0x10   /* timed full receive   */
#define ADTOPTN1_NBLKR  0x40   /* non-blocking receive */
#define ADTOPTN1_DODNR  0x80   /* do DNR calls for UDP */
#define ADTOPTN2_NOSTP  0x40   /* no stripping ll/sep  */
```

```
#define ADTOPTN2_VLIST  0x80    /* vector list          */

    /*-------------------------------------------*/
     /*       CPT DATA TRANSFER SERVICE          */
    /*-------------------------------------------*/
#if defined (IBMC)
 #pragma linkage(t09frecv,OS)
 #pragma linkage(t09fsend,OS)
#else
 #define t09frecv __asm_t09frec
 #define t09fsend __asm_t09fsen
#endif

void t09frecv (), t09fsend ();

#endif
```

PARAMETER	DESCRIPTION
adt_vers	Version Indicates the CPT version number of the argument list used by the calling program. This required field must be set to ADT_VERSN for this release of CPT. Default: None
adt_func	Function code Indicates the function or callable service ID requested by the application program. This field is not set by the application, but is initialized by the TRUE interface stub program. Default: None
adt_token	Data transfer token Specifies a token that represents the connection. A token is obtained from a connection management request. The token is required. Default: None
adt_buffa	User data address Indicates the storage address into which network data is placed. This is a contiguous segment of storage, accessible to the user task. The content of all user data is application-dependent, and is not interpreted by either CPT or by the transport provider. The storage area can be aligned on any boundary convenient for the application program. Default: 0
adt_buffl	User data length Indicates the length (in bytes) of user data in the storage area as identified by the adt_buffa operand. The length is updated when the request is completed to reflect the actual length of user data received. This field must be interpreted on completion to determine the amount of data actually received. If a SEND request is issued with a zero length, an error is detected and the request fails. Default: 0
adt_rtncd	Return code Indicates the return code set by the SEND service. This value indicates the success or failure of the service. Default: 0

200801-024040-200100

PARAMETER	DESCRIPTION
adt_dgncd	Diagnostic code Indicates the diagnostic code received by the SEND service request. This value generally indicates a transport provider return code. Default: 0
adt_stats	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
adt_trace	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
adt_qsend	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
adt_msend	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
adt_qrecv	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
adt_mrecv	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
adt_timeo	RECEIVE timeout value Not used by the SEND service. Default: None
adt_lport	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
adt_rport	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
adt_srvce	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
adt_sepc	Number of sep characters If adt_sepc is not equal to one or two, CPTESEP# will be returned in ADTRTNCD. Default: None
adt_sep1	First or only sep character Default: None
adt_sep2	Second sep character Default: None
adt_laddr	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
adt_raddr	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
adt_lname	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
adt_rname	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
adt_ucntx	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.

PARAMETER	DESCRIPTION
adt_optn2	◆ ADTOPTN2_VLST – Currently for internal use only. ◆ ADTOPTN2_NOSTP – Does not apply to SEND.

adt_optn1	This field specifies data transfer options.
	These are the ADT options that apply to TCP data transfer requests:
	◆ ADTOPTN1_NBLKR – Do not block on a call to the RECEIVE service. Not used by the SEND service.
	◆ ADTOPTN1_TMRCV – Timed full record RECEIVE. Not used by the SEND service.
	◆ ADTOPTN1_TMPRT – Timed partial record RECEIVE. Not used by the SEND service.
	◆ ADTOPTN1_TYPSP – SEP type SEND. These files (along with other required ADT fields) are used to request a SEP type SEND call:
	`adt_optn1 = ADTOPTN1_TYPSP`
	`adt_sepc = 1 OR 2`
	`adt_sep1 = character`
	`adt_sep2 = character if ADTSEP# = 2`
	◆ ADTOPTN1_TYPLL – LL type SEND. These fields (along with other required ADT fields) are used to request a SEP type SEND call:
	`adt_optn1 = ADTOPTN1_TYPLL`
	◆ ADTOPTN1_BLCKS – Block on SEND service call. The default for the SEND service is to return control to the caller as soon as the SEND request has been scheduled with the local transport provider. Option ADTOPTN1_BLCKS will block the return to the caller until the SEND data has been moved to the local transport provider's address space (not the remote TCP). This can provide TCP connection status at the SEND service call time rather than at some later time (next SEND or RECEIVE call).
	◆ ADTOPTN1_DODNR – Does not apply to SEND.
	Note: It is an error to combine any of these SEND service options:
	ADTOPTN1_TYPLL
	ADTOPTN1_TYPSP
	An invalid combination will result in CPTEOPTN being returned in adt_rtncd.
	Default: None

200801-024040-200100

Completion Information

The SEND service completes normally when the data has been both moved from the application program's storage area and forwarded to the transport provider for sending to the connected transport user. A length is returned to the application program, which is set to the amount of data processed.

Normal completion of the SEND service implies nothing in regard to when the data is sent to the peer transport user. This only means that the transport provider has taken custody of the user data and the storage area provided by the application program can be reused. The transport provider generally sends

the buffered data, but this may not occur synchronously with the completion of the SEND service.

On normal return to the application program, the general return code in adt_rtncd is set to zero (CPTIRCOK). The diagnostic code in adt_dgncd is always zero. The length field (adt_buffl) indicates amount of data processed.

If the SEND service completes abnormally, some or no user data may have been sent to the peer transport user. The general return code in adt_rtncd, and the diagnostic code in adt_dgncd, indicate the nature of the failure. The diagnostic code (adt_dgncd) generally contains a specific code that is generated by the transport provider.

Return Codes

The SEND service return and diagnostic codes indicate the result of the execution. These values are in the adt_rtncd and adt_dgncd within the ADT. The diagnostic code is optional and generally indicates a transport provider return code. Read **Appendix C – MESSAGES AND CODES** in the ***CICS Programmer's Toolkit Installation and Administration Guide*** for the return code cross reference table.

This table describes the return codes for the SEND service:

RETURN	DIAGNOSTIC CODE	DESCRIPTION
CPTIRCOK		Successful.
CPTEVERS		Control block version number is not supported.
CPTETOKN		Specified token is not valid.
CPTEBUFF		Buffer address and/or length is invalid.
CPTEPRGE	Yes	CPT Interface is terminating.
CPTENAPI	Yes	Transport provider API is not available.
CPTETERM	Yes	Environment is being terminated.
CPTERLSE	Yes	Release indication.
CPTEDISC	Yes	Disconnect indication.
CPTEINTG	Yes	Transport provider API integrity error.
CPTEENVR	Yes	Transport provider API environment error.
CPTEFRMT	Yes	Transport provider API format error.
CPTEPROC	Yes	Transport provider API procedure error.
CPTABEND		Abnormal exception occurred.
CPTEOTHR	Yes	An undefined exception occurred.

Usage Information

The SEND service sends normal data as output through a CPT connection. The data is part of a byte stream being sent over a connection (TCP).

Data is moved from the application program's storage area to storage areas maintained by the transport provider, is packetized, and is sent to the connection transport user. Logical boundaries are not preserved in the data stream. The data is delivered to the peer transport user in the precise order in which it was sent. However, this data may be fragmented.

Data is not necessarily packetized and sent by the transport provider each time a SEND service is issued, nor is it sent when a buffer boundary is indicated. The transport provider may intentionally delay sending data as the result of performance optimization or congestion avoidance algorithms. Normally, data generated by the application is forwarded when it is sent in a continuous flow.

The SEND service request is a synchronous operation that may require the application to be blocked. The SEND queue (acm_qsend) and buffer (acm_msend) sizes, as specified during connection initialization, control output queuing and buffering. The queue size represents the number of uncompleted SEND requests that can be pending to the transport provider. The default value is generally sufficient to support most applications. The transport provider normally completes a SEND request within a reasonable amount of time, but a congested network may cause the transport provider delays in completing requests and cause the application to be blocked.

Additionally, for an application that issues multiple SEND requests contiguously, users should consider increasing the default queue size. The buffer size represents the maximum number of user data bytes that can be transferred by the application in a single SEND request to the transport provider. This value is application-dependent. A small value causes the SEND service to issue multiple transport provider SEND requests. Multiple transport provider SEND requests do not present a problem, but have an effect of reducing the queue size and can cause the application to be blocked. A large buffer value can waste application storage.

The queue and buffer size values are specified during connection initialization and can be modified on return. An application that is dependent on these values should validate the requested values, compared with values returned within the ACM. The values are modified if the transport provider site administrator has configured limits and the application request exceeds those values. If the requested values are modified, the user should verify site definition statements for API transport services.

The version number (adt_vers) indicates the CPT release level in which this user application program is written. This required field must be set to ADT_VERSN and is validated by the SEND service before processing the request.

The function code (adt_func) indicates the CPT callable service ID. The field is not initialized by a user application program and has little value to the application except for dump analysis. The function code identifies and maps an argument list with the error or trace log and dump analysis.

200801-024040-200100

The token (`adt_token`) indicates the connection that is to transmit data. This is a required field and is validated by the `SEND` service before processing the request.

The data buffer address field `adt_buffa` is a full word. The application program is responsible for assuring that the residency mode of data areas it manages (for example, argument lists) is compatible with the addressing mode. The transport provider performs consistency checks on the addressing mode whenever a service request is issued. However, unpredictable results may occur before the transport provider performs this check.

The data buffer length is indicated by the `adt_buffl` field. This is a full word positive integer. The data buffer length field should be less than or equal to the maximum send buffer values. However, if the data buffer length is greater than the maximum send buffer, the `SEND` service fragments the user data into multiple transport provider requests. The `adt_buffl` is updated on return from the `SEND` service with a value that indicates the number of bytes processed.

The `adt_optn1` field specifies `SEND` processing control options and provides a mechanism for event notification on return to application program.

20 024040-200100

Example:

This example establishes a connection and sends data to the remote host. The token is loaded from the ACM and used by all of the following CPT service requests. The `adt_rtncd` is checked to determine `SEND` service request completion status.

```
#include  <t09ksadt.h>
#include  <t09ksacm.h>
#include  <t09ksrcs.h>

void main()
{
    struct adt_stru
        cpt_adt = {2, 0, 0, NULL, NULL, 0, 0, 0, 0};

    char   message[256];

    /*
    *  CPT Connection Management service request
    */

    cpt_adt.adt_token = cpt_acm.acm_token;

    /*
    *  Data Processing Routine
    */

    while (data)
    {
        cpt_adt.adt_buffa = &msgarea;
        cpt_adt.adt_buffl = msglen;
        t09fsend (&cpt_adt);
        if (cpt_adt.adt_rtncd != 0)
        {
            /*
            * process CPT SEND service error and terminate transaction;
            */
        }
    }
    /*
    *  CPT Release Connection
    */

    /*
    *  Terminate Transaction
    */

    End:
    EXEC CICS RETURN;
}
```

200801-024040-200100

SENDTO

The SENDTO service is provided to allow connectionless client and server applications to be developed. This service is UDP only. The SENDTO service provides two basic functions:

- Establish a UDP client endpoint represented by a new token and send a datagram to a remote UDP server. This function is indicated to the SENDTO service by passing an ADTTOKEN equal to zero. SENDTO will then create all the internal control blocks and the SENDTO buffer queue. Even though the RCVFROM buffer queue will not be allocated for this endpoint (token) until the RCVFROM service is called, the RCVFROM buffer size and number must be specified at this time because they are negotiated with the transport provider and recorded in the internal CPT control blocks at endpoint creation time. Upon return from the SENDTO service adt_token will contain the token value to be passed to subsequent SENDTO and RCVFROM service calls.
- Send a datagram at a previously established UDP endpoint represented by an existing token. This functionality makes the SENDTO service call just a data transfer call that can be used by a client or server application. The SENDTO buffer queue is only allocated upon the first call to the SENDTO service whether adt_token is equal to zero or not.

UDP tokens created with the RCVFROM or SENDTO services cannot be passed to the TCP only services, CONNECT, LISTEN, SEND, and RECEIVE. All other CPT service calls are available to UDP applications.

This table describes the arguments for the SEND service:

INCLUDE STATEMENT	STRUCT NAME	SIZE	CREATED BY
t09ksadt.h	adt_stru	28 (X'1C')	User application

Structure

This is the structure of the SEND service in C language:

```
#ifndef __t09ksadt__
#define __t09ksadt__
 /* t09ksadt.h --  Declare CPT-dependent structures     */

 /*     ACCCSECT UPDTID=A1000000                        */

 /*                                                     */
 /*  Define Data Transfer (ADT) Structure               */
 /*                                                     */

typedef struct
     {
     short adt_vers     ;  /* adt block version number  */
     short adt_func     ;  /* Request function type     */
     int  *adt_token    ;  /* Data transfer token       */
     void *adt_buffa    ;  /* Data buffer address       */
```

```
    int   adt_buffl    ;  /* Buffer size/data length   */
     int   adt_rtncd     ;  /* Return code              */
     int   adt_dgncd     ;  /* Diagnostic code          */
     int   adt_stats     ;  /* Statistics options       */
     int   adt_trace     ;  /* Trace options            */
     int   adt_qsend     ;  /* Queue size - send        */
     int   adt_msend     ;  /* Buffer size - send       */
     int   adt_qrecv     ;  /* Queue size - recv        */
     int   adt_mrecv     ;  /* Buffer size - recv       */
     int   adt_timeo     ;  /* Timed receive - secs     */
     int   adt_rsvd1     ;  /* reserved (in use)        */
     int   adt_rsvd2     ;  /* reserved (in use)        */
     int   adt_rsvd3     ;  /* reserved (in use)        */
     int   adt_rsvd4     ;  /* reserved (in use)        */
     short adt_lport     ;  /* Local port               */
     short adt_rport     ;  /* Remote port              */
     short adt_rsvd5     ;  /* reserved (in use)        */
     char  adt_srvce [36];  /* Service name             */
     short adt_sepc      ;  /* # of sep characters      */
     char  adt_sep1      ;  /* 1st/only sep character   */
     char  adt_sep2      ;  /* 2nd sep character        */
     short adt_rsvd6     ;  /* reserved                 */
     int   adt_laddr     ;  /* Local host ip address    */
     int   adt_raddr     ;  /* Remote host ip address   */
     char  adt_lname[256];  /* Local host name          */
     char  adt_rname[256];  /* Remote host name         */
     int   adt_ucntx     ;  /* User context field       */
     char  adt_optn4     ;  /* Special options - unused */
     char  adt_optn3     ;  /* Special options - unused */
     char  adt_optn2     ;  /* Special options - byte 2 */
     char  adt_optn1     ;  /* Special options - byte 1 */
     } adt_stru;

     /*-------------------------------------------*/
      /*       CONTROL BLOCK VERSION NUMBER       */
     /*-------------------------------------------*/
#define ADT_VERSN       2

     /*-------------------------------------------*/
      /*       RETURN CODE AT END-OF-FILE         */
     /*-------------------------------------------*/
#define NO_MORE_DATA   65  /* Remote Requests Orderly Release */

     /*-------------------------------------------*/
      /*            STATISTICS OPTIONS            */
     /*-------------------------------------------*/
 #define ADTSTATS_CONN    0x01 /* CONNECTION STATISTICS */
 #define ADTSTATS_TERM    0x02 /* TERMINATION STATISTICS*/

     /*-------------------------------------------*/
      /*            TRACE LOG REQUESTS            */
     /*-------------------------------------------*/
 #define ADTTRAC1_NTRY   0x01   /*    ENTRY POINTS    */
 #define ADTTRAC1_ARGS   0x02   /*     ARGUMENTS      */
 #define ADTTRAC1_RECV   0x04   /*       TRECV        */
 #define ADTTRAC1_SEND   0x08   /*       TSEND        */
 #define ADTTRAC1_TERM   0x10   /*    TERMINATION     */
 #define ADTTRAC1_PASS   0x20   /*     GIVE/TAKE      */
 #define ADTTRAC1_CLSE   0x40   /*      CLOSE         */
```

```
#define ADTTRAC1_TERR   0x80    /*     TPLERRORS      */
#define ADTTRAC2_TOKN   0x01    /*      TOKENS        */
#define ADTTRAC2_TPL    0x02    /*        TPL         */
#define ADTTRAC2_RLSE   0x04    /*      RELEASE       */
#define ADTTRAC2_STOR   0x08    /*      STORAGE       */

    /*-------------------------------------------*/
     /*            SPECIAL OPTIONS                */
    /*-------------------------------------------*/
#define ADTOPTN1_TYPSP  0x01   /* record by separators */
#define ADTOPTN1_TYPLL  0x02   /* record by ll prefix  */
#define ADTOPTN1_BLCKS  0x04    /* blocking send       */
#define ADTOPTN1_TMPRT  0x08   /* timed partial receive*/
#define ADTOPTN1_TMRCV  0x10   /* timed full receive   */
#define ADTOPTN1_NBLKR  0x40   /* non-blocking receive */
#define ADTOPTN1_DODNR  0x80   /* do DNR calls for UDP */
#define ADTOPTN2_NOSTP  0x40   /* no stripping ll/sep  */
#define ADTOPTN2_VLIST  0x80    /* vector list         */

    /*-------------------------------------------*/
     /*       CPT DATA TRANSFER SERVICE           */
    /*-------------------------------------------*/
#if defined (IBMC)
 #pragma linkage(t09frecv,OS)
 #pragma linkage(t09fsend,OS)
#else
 #define t09frecv __asm_t09frec
 #define t09fsend __asm_t09fsen
#endif

void t09frecv (), t09fsend ();

#endif
```

PARAMETER	DESCRIPTION
adt_vers	Version Indicates the CPT version number of the argument list used by the calling program. This required field must be set to ADT_VERSN for this release of CPT. Default: None
adt_func	Function code Indicates the function or callable service ID requested by the application program. This field should not be set by the application, but rather is initialized by the true interface stub. Default: None
adt_token	Data transfer token Specifies a token that represents a UDP endpoint. If the ADT is being passed in a call to either the RCVFROM or SENDTO service, the token can be zero, indicating to either service, to first create a token before sending or receiving a datagram. If the token is not zero, it must be a token created previously by either the RCVFROM or SENDTO service. It is not necessary or efficient to create a token every time a CICS transaction calls the UDP data transfer services. It is an error to pass a TCP token to the UDP data transfer service routines, RCVFROM and SENDTO. Conversely, it is an error to pass a UDP token to the TCP data transfer routines, RECEIVE and SEND.

PARAMETER	DESCRIPTION
adt_buffa	User data address Indicates the storage address from which the UDP datagram will be sent (SENDTO service). This is a contiguous segment of storage accessible to the user task. The content of all user data is application dependent, and not interpreted by either CPT nor the transport provider. The storage area can be aligned on any boundary convenient for the application program. Default: 0
adt_buffl	User data length Indicates the length (in bytes) of the buffer specified in adt_buffa which is to be sent (SENDTO service). It is an error to call the SENDTO service with an adt_buffl of zero. Upon return to the caller, adt_buffl reflects the number of bytes actually sent (generally the number requested). Indicates the return code set by the SENDTO service. Default: 0
adt_rtncd	Return code Indicates the return code set by the CLOSE service. This value is also returned in register 15 and indicates the success or failure of the service. Default: 0
adt_dgncd	Diagnostic code Indicates the diagnostics code set by the SENDTO service. This value generally indicates a transport provider return code. Default: 0
adt_stats	adtstats_conn | adtstats_term Specifies statistics logging options for the application program. The facility can be used for debugging and tuning during development. ◆ adtstats_conn – Specifies that a message(s) be generated on establishing either a listen service or a data transfer connection. These messages are generated by the CPT LISTEN and CONNECT services. The message numbers associated with this option are CPT802I and CPT803I. ◆ adtstats_term – Specifies that a message(s) be generated on terminating an established connection. These messages are generated by the CPT CLOSE service. The message numbers associated with this option are CPT807I, CPT808I, and CPT809I. Default: 0 (no statistics logging)

200801-024040-200100

PARAMETER	DESCRIPTION
adt_trac1	Specifies trace logging options for the application program. The facility can be used for debugging during development. ◆ adttrac1_ntry – Specifies that a message be generated on entry to a CPT service routine. The message is generated by all CPT service routines. The message number associated with this option is CPT901I. ◆ adttrac1_args – Specifies that a hex dump of the caller's arguments be generated on entry and exit of an CPT service routine. Messages are generated by the corresponding service routine. Message numbers associated with this option are CPT902I, CPT903I, CPT904I, CPT911I, CPT912I, CPT921I,CPT922I, CPT924I, CPT925I, CPT930I, and CPT932I. ◆ adttrac1_recv – Specifies that a hex dump of the transport provider's input (RECEIVE) data be logged. Messages are generated by the CPT RECV service. The message number associated with this option is CPT913I. ◆ adttrac1_send – Specifies that a hex dump of the transport provider's output (SEND) data be logged. Messages are generated by the CPT SEND service. The message number associated with this option is CPT914I. ◆ adttrac1_term – Specifies that a message be generated on termination of an CPT service routine. Messages are generated by all CPT service routines. The message number associated with this option is CPT908I. ◆ adttrac1_pass – Specifies that a hex dump of resources related to a passed connection be logged. Messages are generated by the CPT GIVE and TAKE services. The message numbers associated with this option are CPT919I and CPT920I. ◆ adttrac1_clse – Specifies that a hex dump of resources related to a CLOSE processing of a connection be logged. Message are generated by the CPT CLOSE service. The message number associated with this option is CPT906I. ◆ adttrac1_terr – Specifies that a hex dump of a transport provider API parameter list that fails successful completion be logged. The message number associated with this option is CPT400I.
adt_trac2	◆ adttrac2_tokn – Specifies that a hex dump of the connection token be logged. Messages are generated by all services routines on entry. The message number associated with this option is CPT909I. ◆ adttrac2_tpl – Specifies that a hex dump of a transport provider API parameter list be logged. The message numbers associated with this option are CPT917I and CPT935I. ◆ adttrac2_rlse – Specifies that a message indicating a transport provider release indication be logged. Messages are generated by various CPT service routines. The message number associated with this option is CPT414I. ◆ adttrac2_stor – Specifies that a hex dump of storage management arguments be logged. Messages are generated by various CPT service routines. The message number associated with this option is CPT928I and CPT929I. Default: 0 (no trace logging)
adt_qsend	API send queue size Specifies the maximum number of uncompleted SENDTO requests that can be queued by the application to the transport provider (API). This value lets applications control output processing and can affect throughput rates. The value is negotiated with the transport provider and may be modified by the transport provider. The total allocation for output processing is the product of the SENDTO queue and buffer size values and cannot exceed 61K. Default: 4
adt_msend	API send buffer size Specifies the maximum number of user data bytes that can be transferred by the application in a single SENDTO request to the transport provider (API). This value lets applications control output processing and can affect throughput rates. The value is negotiated with and may be modified by the transport provider. The total allocation for output processing is the product of the SENDTO queue and buffer size values and cannot exceed 61K. Default: 1024

PARAMETER	DESCRIPTION
adt_qrecv	API receive queue size Specifies the maximum number of uncompleted RCVFROM requests that can be queued by the application to the transport provider (API). This value lets applications control input processing and can affect throughput rates. The value is negotiated with and may be modified by the transport provider. The total allocation for input processing is the product of the RCVFROM queue and buffer size values and cannot exceed 61K. Default: 4
adt_timeo	RECEIVE timeout value Not used by the SENDTO service Default: 0
adt_lport	Local well-known service port. Indicates the local transport layer port that the calling application will be sending (SENDTO) UDP datagrams from. If the SENDTO service creates the token, this port number is assigned by the transport layer and returned to the caller. If the RCVFROM service creates the token, this is the well-known port requested by the caller. This field is an unsigned positive integer with a maximum value of 65,534. The value must be unique for each server application. Default: None
adt_rport	Remote port Indicates the remote transport layer port on which the incoming datagram originated. This value is returned to the caller of the RCVFROM service and may be different for each datagram received. This field is an unsigned positive integer with a maximum value of 65,534. Default: None
adt_srvce	Transport layer service name (used when adt_token=0) Indicates the local transport layer service name. This string is mapped into a transport layer address or port, through internal DNR. The resolved value is the well-known port to which the client application sends (SENDTO) UDP datagrams. This field has a maximum length of 36 bytes. The value must be unique for each server application. This field is optional and is not modified by the RCVFROM service. Default: None
adt_sepc	Number of separator characters for option ADTOPTN1_TYPSP. Not used in the SENDTO service. Default: None
adt_sep1	First or only separator character for option ADTOPTN1_TYPSP. Not used in the SENDTO service. Default: None
adt_sep2	Second character or separator sequence for option ADTOPTN1_TYPSP. Not used in the SENDTO service. Default: None
adt_laddr	Local host IP address Indicates the local host internet address. This field is an unsigned four-byte integer value. The local host internet address is returned to the caller of the RCVFROM Default: None
adt_raddr	Remote host IP address Indicates the remote host internet address of the sender of the incoming UDP datagram. This value is returned to the caller of the SENDTO service and may be different for each datagram received. This field is an unsigned four-byte integer value. Default: None

200801-024040-200100

PARAMETER	DESCRIPTION
adt_lname	Local host name Indicates the local host internet name. This field is a 255-byte character string that is padded with blanks. The local host internet name is returned to the caller of the SENDTO service.
adt_rname	Remote host name Indicates the remote host internet name. This field is a 255-byte character string that is padded with blanks. It is only resolved through internal DNR calls and returned to the caller of the UDP data transfer service routines (RCVFROM and SENDTO) if the adt_optn1 flag, ADTOPTN1_DODNR is specified. This is to prevent the DNR call overhead on every UDP data transfer call. Default: None
adt_ucntx	One word of user context Specifies one arbitrary word of user context to be associated with the connection. The information provided is not interpreted by CPT, and is merely saved with other connection information. Default: 0 (no user context)
adt_optn2	◆ ADTOPTN2_VLST – Currently for internal use only. ◆ ADTOPTN2_NOSTP – Does not apply to SENDTO.
adt_optn1	Specifies data transfer options These are the ADT options that apply to UDP data transfer requests: ◆ ADTOPTN1_DODNR – Execute internal DNR calls during UDP data transfer service routine calls (RCVFROM and SENDTO) to resolve remote IP addresses into IP names in the ADTRNAME field. ◆ ADTOPTN1_NBLKR – Do not block on a call to the RCVFROM service (not used by the SENDTO service). Default: None These options can be toggled on every UDP data transfer call even if the caller is using the same token. 1

Network Considerations

The ADT is a common data structure used for both client and server UDP applications. There are common and unique values specified for a particular service request.

This table describes network considerations for C API:

NAME	SERVER CONDITIONS FOR RCVFROM	CLIENT CONDITIONS FOR SENDTO
adt_lport	Local server well-known port selected by user application.	Local assigned transport provider port returned to user application.
adt_rport	Remote client transport provider port returned to user by user application.	Remote server transport provider well-known port selected by user application.
adt_srvce	Local server trans port provider service name selected by user application.	Remote server transport provider service name selected by user application.
adt_raddr	Remote IP host address returned to user application.	Remote IP host address selected by or returned to user application. The client must specify this field or adt_rname.
adt_lname	Local IP host name returned to user application.	Local IP host name returned to user application.
adt_rname	Remote IP host name returned to user application only if ADTOPTN1_DODNR is specified in adt_optn1.	Remote IP host name selected by or returned to the user application. The client must specify this field or adt_raddr. If adt_raddr is used, adt_rname will only be returned if ADTOPTN1_DODNR is specified in adt_optn1.

200801-024040-200100

Return Codes

The SEND service return and diagnostic codes indicate the result of the execution. These values are in the adt_rtncd and adt_dgncd within the ADT. The diagnostic code is optional and generally indicates a transport provider return code. Read **Appendix C – MESSAGES AND CODES** in the ***CICS Programmer's Toolkit Installation and Administration Guide*** for the return code cross reference table.

This table describes the return codes for the SEND service:

RETURN	DIAGNOSTIC CODE	DESCRIPTION
CPTIRCOK		Successful.
CPTEVERS		Control block version number is not supported.
CPTETOKN		Specified token is not valid.
CPTEBUFF		Buffer address and/or length is invalid.
CPTEPRGE	Yes	CPT Interface is terminating.
CPTENAPI	Yes	Transport provider API is not available.
CPTETERM	Yes	Environment is being terminated.
CPTERLSE	Yes	Release indication.
CPTEDISC	Yes	Disconnect indication.
CPTEINTG	Yes	Transport provider API integrity error.
CPTEENVR	Yes	Transport provider API environment error.
CPTEFRMT	Yes	Transport provider API format error.
CPTEPROC	Yes	Transport provider API procedure error.
CPTABEND		Abnormal exception occurred.
CPTEOTHR	Yes	An undefined exception occurred.

Example:

This example sends data to a remote host. The `adt_rtncd` is checked to determine `SENDTO` service request completion status.

```
#include  <t09ksadt.h>
#include  <t09ksrcs.h>

void main()
{
    struct adt_stru
        cpt_adt = {ADT_VERSN, 0, 0, NULL, NULL, 0, 0, 0, 0};

    char   message[] = "Sample CPT C program message";

    /*
    *  Identify Service
    */

    cpt_adt.adt_rport = 1680;
    memcpy (cpt_adt.adt_rname, "127.0.0.1",9);

    /*
    *  Data Processing Routine
    */

    while (data)
    {
        cpt_adt.adt_buffa = &message;
        cpt_adt.adt_buffl = sizeof(message);
        t09fsnto (&cpt_adt);
        if (cpt_adt.adt_rtncd != 0)
        {
            /*
            *  process CPT SEND service error and terminate transaction;
            */
        }
    }
    /*
    *  CPT Terminate Endpoint
    */

    /*
    *  Terminate Transaction
    */

    End:
    EXEC CICS RETURN;
}
```

TAKE

The TAKE service establishes ownership of a connection and associated internal CPT resources. The TAKE service is optional and does not affect an active connection nor does it issue any transport provider requests. This service affects CPT TRUE management routines scheduled on the user's behalf during task termination.

To invoke the TAKE service, a user application is required to first build an AFM and then to issue a call to the TAKE routine. The only valid and required arguments are the AFM version number and the connection token. On completion, a return code is set to indicate success or failure of the request.

This table describes the arguments for the TAKE service:

INCLUDE STATEMENT	STRUCT NAME	SIZE	CREATED BY
t09ksafm.h	afm_stru	34 (X'22')	user application

Structure

This is the structure of the TAKE service in C language:

```
#ifndef __t09ksafm__
#define __t09ksafm__
 /* t09ksafm.h --  Declare CPT-dependent structures     */

 /*     ACCCSECT UPDTID=A1000000                         */

 /*                                                      */
 /*  Define Facility Management (AFM) Structure          */
 /*                                                      */

typedef struct
     {
     short afm_vers     ;  /* AFM block version number  */
     short afm_func     ;  /* Request function type     */
     int  *afm_token    ;  /* Data Transfer token       */
     void *afm_buffa    ;  /* Reserved                  */
     int   afm_buff1    ;  /* Reserved                  */
     int   afm_rtncd    ;  /* Return code               */
     int   afm_dgncd    ;  /* Diagnostic code           */
     int   afm_rsvd1    ;  /* reserved (in use)         */
     short afm_rsvd2    ;  /* reserved (in use)         */
     } afm_stru;

     /*-----------------------------------------------*/
      /*       CONTROL BLOCK VERSION NUMBER       */
     /*-----------------------------------------------*/
#define AFM_VERSN       2

     /*-----------------------------------------------*/
      /*       CPT FACILITY MANAGEMENT SERVICE     */
     /*-----------------------------------------------*/
 #if defined (IBMC)
  #pragma linkage(t09fgive,OS)
  #pragma linkage(t09ftake,OS)
 #else
  #define t09fgive __asm_t09fgiv
```

```
 #define t09ftake __asm_t09ftak
#endif

void t09fgive (), t09ftake ();

#endif
```

PARAMETER	DESCRIPTION
afm_vers	Version Indicates the CPT version number of the argument list used by the calling program. This required field must be set to AFM_VERSN for this release of CPT. Default: None
afm_func	Function code Indicates the function or callable service ID requested by the application program. This field is not set by the application, but is initialized by the TRUE interface stub program. Default: None
afm_token	Connection token Specifies a token that represents the connection. A token is obtained from a connection management request. The token is required. Default: None
afm_rtncd	Return code Indicates the return code set by the TAKE service. This value indicates the success or failure of the service. Default: 0
afm_dgncd	Diagnostic code Indicates the diagnostic code received by the TAKE service for a transport provider request. The TAKE service does not issue transport provider requests; hence, it never sets the diagnostic code. Default: 0

200801-024040-200100

Completion Information

The TAKE service completes normally when all resources associated with this connection are processed.

On normal return to the application program, the general return code in afm_rtncd is set to zero (CPTIRCOK). The diagnostic code in afm_dgncd is always zero.

If the TAKE service completes abnormally, then some resources associated with this connection cannot be successfully transferred from one task to another. The general return code in afm_rtncd and the diagnostic code in afm_dgncd indicate the nature of the failure. The diagnostic code (afm_dgncd) is not used by the TAKE service and no information is returned.

Return Codes

The TAKE service return code indicates the result of the execution. These values are in the afm_rtncd within the ADT. Read **Appendix C – MESSAGES AND CODES** in the ***CICS Programmer's Toolkit Installation and Administration Guide*** for the return code cross reference table.

This table describes the TAKE service return codes:

RETURN	DIAGNOSTIC CODE	DESCRIPTION
CPTIRCOK		Successful.
CPTEVERS		Control block version number is not supported.
CPTETOKN		Specified token is not valid.
CPTENAPI		Transport provider API is not available.
CPTABEND		Abnormal exception occurred.
CPTEOTHR		An undefined exception occurred.

Usage Information

The TAKE service acquires ownership of a connection from one task to another. This service is non-blocking and does not effect any pending transport provider data transfer requests. The association established by the TAKE service lets the CPT properly manage resources during task termination. This ability to GIVE and TAKE ownership of connections offers the user a range of programming options, while still providing CPT with resource management capabilities.

The TAKE service requires the application to set the AFM version number and token fields. No other AFM fields are referenced.

When a connection is established there are internal CPT resources associated with that connection. CPT is responsible for proper clean-up of those resources on task or transaction termination. These resources include storage allocated by CPT, the API, and the transport provider storage.

A server application is a good example of how the TAKE service benefits a user application. A listening task issues the GIVE service and starts a new transaction to handle data transfer. The data transfer transaction then takes the connection. This sequence prevents a connection from being closed (implicitly by the CPT task termination exit) if the server application terminates. However, if the data transfer transaction terminates without issuing an explicit close (CPT CLOSE service), an implicit close is scheduled and resource management is handled by the CPT task termination exit.

Additionally, an implicit TAKE facility is implemented with the SEND, RECEIVE, and TRANSLATE services. The task to issue a SEND, RECEIVE, or TRANSLATE service obtains control of the connection and associated resources. The advantage is that the TAKE service is optional. Even though TAKE can be implicit and therefore optional, Interlink recommends that you issue TAKE to

avoid having a GIVE connection not associated with any transactions. Ownership of a connection and resources provide for clean-up processing during abnormal termination.

The version number (afm_vers) indicates the CPT release level in which this user application program is written. This required field must be set to AFM_VERSN and is validated by the TAKE service before processing the request.

The function code (afm_func) indicates the CPT callable service ID. The field is not initialized by a user application program and has little value to the application except for dump analysis. The function code identifies and maps an argument list with the error or trace log and dump analysis.

The token (afm_token) indicates the connection and internal resources that are to be processed by the TAKE service. This is a required field and is validated by the TAKE service before it processes the request.

The afm_optns field specifies TAKE service processing control options, and provides a mechanism for event notification on return to the application program. Currently, this field is not used by application programs.

200801-024040-200100

Example:

In this example a connection is established by another task and is passed to a data processing transaction. This application retrieves the ACM and takes ownership of the connection and resources. The token is loaded from the ACM and used by all of the following CPT service requests. The `afm_rtncd` is checked to determine `TAKE` service request completion status.

```
#include  <t09ksafm.h>
#include  <t09ksacm.h>
#include  <t09ksrcs.h>

void main()
{
    struct afm_stru
        cpt_afm = {2, 0, 0, NULL, NULL, 0, 0, 0, 0};

    CICS RETRIEVE SET(cpt_acm) LENGTH(acmleng);

    cpt_afm.afm_token = cpt_acm.acm_token;

    t09ftake (&cpt_afm);
    if (cpt_afm.afm_rtncd != 0)
    {
        /*
        * process CPT TAKE service error and terminate transaction;
        */
    }
    while (data)
    {
        /*
        * Application and CPT Data Transfer (SEND/RECEIVE) processing
        */
    }
    /*
    * CPT Release Connection
    */

    /*
    * Terminate Transaction
    */

    End:
    EXEC CICS RETURN;

}
```


20-024040-200100

Example:

In this example a data processing transaction retrieves the ACM and performs some required data transfer operation. The token is loaded from the ACM and is used by for CPT service requests. The difference between this example and the previous example is that no TAKE services is explicitly issued by the application. The CPT data transfer services, SEND and RECEIVE, implement the TAKE service internally, so an explicit TAKE service request can be omitted.

```
#include  <t09ksacm.h>
#include  <t09ksadt.h>
#include  <t09ksrcs.h>

void main()
{
    struct adt_stru
        cpt_adt = {2, 0, 0, NULL, NULL, 0, 0, 0, 0};
    CICS RETRIEVE SET(cpt_acm) LENGTH(acmleng);
    cpt_adt.adt_token = cpt_acm.acm_token;
    while (data)
    {
        /*
        * Application and CPT Data Transfer (SEND/RECEIVE) processing
        */
    }
    /*
    *  CPT Release Connection
    */

    /*
    *  Terminate Transaction
    */

    End:
    EXEC CICS RETURN;
}
```

200801-024040-200100

TRANSLATE

The TRANSLATE service translates data between EBCDIC and ASCII character sets. CPT is customized with a default translation table; however, applications have the ability to override the default. The TRANSLATE service does not affect an active connection nor issue any transport provider requests.

To invoke the TRANSLATE service, a user application is required to first build an Argument for Data Translation (AXL) and then to issue a call to the TRANSLATE routine. The AXL is required to contain the version number, connection token, user buffer address and length, and type or direction of translation requested. Additional arguments for application specific translation tables are supported. On completion of the TRANSLATE service, the buffer contents are converted into the corresponding characters and a return code is generated indicating the status of the request.

This table describes the arguments for the TRANSLATE service:

INCLUDE STATEMENT	STRUCT NAME	SIZE	CREATED BY
t09ksaxl.h	axl_stru	32 (X'20')	User application

Structure

This is the structure of the TRANSLATE service in C language:

```
 #ifndef __t09ksaxl__
#define __t09ksaxl__
 /* t09ksaxl.h --  Declare CPT-dependent structures    */

 /*     ACCCSECT UPDTID=A2002007                        */

 /*                                                     */
 /*  Define Data Translation (AXL) Structure            */
 /*                                                     */

 typedef struct
     {
    short axl_vers     ;  /* axl block version number  */
    short axl_func     ;  /* Request function type     */
    int  *axl_token    ;  /* Data transfer token       */
    void *axl_saddr    ;  /* Text buffer address       */
    int   axl_sleng    ;  /* Text data length          */
    int   axl_rtncd    ;  /* Return code               */
    int   axl_dgncd    ;  /* Diagnostic code           */
    char  axl_mfill    ;  /* Reserved translation mode */
    char  axl_xmode    ;  /* Translation mode          */
    char  axl_tfill    ;  /* Reserved translation type */
    char  axl_xtype    ;  /* Translation type options  */
    void *axl_table    ;  /* User translation table    */
     } axl_stru;

     /*-------------------------------------------*/
      /*       CONTROL BLOCK VERSION NUMBER       */
     /*-------------------------------------------*/
```

```
#define AXL_VERSN       2

     /*-------------------------------------------*/
      /*          CHARACTER SET MODE             */
     /*-------------------------------------------*/
#define AXLXMODE_SBCS  0x00 /*      SINGLE BYTE C.S.     */
#define AXLXMODE_DBCS  0x01 /*      DOUBLE BYTE C.S.     */
#define AXLXMODE_MIXD  0x02 /*      MIXED SBCS/DBCS      */
#define AXLXMODE_NUMS  0x04 /*       NUMBER SET          */

     /*-------------------------------------------*/
      /*       TRANSLATION TYPE REQUEST          */
     /*-------------------------------------------*/
#define AXLXTYPE_ATOE  0x01 /*     ASCII-TO-EBCDIC       */
#define AXLXTYPE_ETOA  0x02 /*     EBCDIC-TO-ASCII       */
#define AXLXTYPE_AUPC  0x04 /*     ASCII-TO-UPPER CASE   */
#define AXLXTYPE_EUPC  0x08 /*     EBCDIC-TO-UPPER CASE  */

     /*-------------------------------------------*/
      /*       CPT DATA TRANSLATION SERVICE       */
     /*-------------------------------------------*/
 #if defined (IBMC)
  #pragma linkage(t09fxlat,OS)
 #else
  #define t09fxlat __asm_t09fxla
 #endif

 void t09fxlat ();

#endif
```

PARAMETER	DESCRIPTION
axl_vers	Version Indicates the CPT version number of the argument used by the calling program. This required field must be set to AXL_VERSN for this release of CPT. Default: None
axl_func	Function code Indicates the function or callable service ID requested by the application program. This field is not set by the application, but is initialized by the TRUE interface stub program. Default: None
axl_token	Connection token Specifies a token that represents a TCP connection or UPD endpoint. Default: None
axl_saddr	Source text buffer address Indicates the address of the user data buffer to be translated. This required field is a contiguous segment of storage accessible to the user task. The storage area can be aligned on any boundary convenient to the application program. Default: 0

200801-024040-200100

PARAMETER	DESCRIPTION
axl_sleng	Source text buffer length Indicates the length (in bytes) of user data buffer in the storage area, as identified by the axl_saddr field. This is a required field. A zero value causes the request to fail. Default: 0
axl_rtncd	Return code Indicates the return code set by the TRANSLATE service. This value indicates the success or failure of the service. Default: 0
axl_dgncd	Diagnostic code Indicates the diagnostic code set by the service request. This value specifies a unique number associated with the return code and identifies the translation error. Default: 0
axl_mfill	Reserved
axl_xmode	Specifies TRANSLATE service translation mode or character set. ◆ XLXMODE_SBCS – Indicates single-byte character set translation. ◆ XLXMODE_DBCS – Indicates double-byte character set translation. This option is currently not supported. ◆ XLXMODE_MIXD – Indicates mixed mode character set translation. This mode specifies single- and double-byte translation. This option is currently not supported. ◆ AXLXMODE_NUMS – Indicates numeric set translation. This option is currently not supported. Default: AXLXMODE_SBCS
axl_mfill	Reserved
axl_xtype	Specifies TRANSLATE service translation type or direction. ◆ AXLXTYPE_ATOE – Indicates ASCII to EBCDIC translation. ◆ AXLXTYPE_ETOA – Indicates EBCDIC to ASCII translation. ◆ AXLXTYPE_AUPC – Indicates ASCII to uppercase ASCII translation. ◆ AXLXTYPE_EUPC – Indicates EBCDIC to uppercase EBCDIC translation. Default: None
axl_table	Address of user translation table. Default: None

Completion Information

The TRANSLATE service completes normally when the data is translated into the corresponding character set representation.

On normal return to the application program, the general return code in the axl_rtncd field is set to zero (CPTIRCOK). The diagnostic code in general the axl_dgncd field is set to zero.

If the TRANSLATE service completes abnormally, an error associated with translation occurred. The general return code in the axl_rtncd field and the diagnostic code in the axl_dgncd field indicate the nature of the failure.

Return Codes

The TRANSLATE service return and diagnostic codes indicate the result of the execution. These values are in the axl_rtncd and axl_dgncd fields within the AXL. Read **Appendix C – MESSAGES AND CODES** in the ***CICS Programmer's Toolkit Installation and Administration Guide*** for the return code cross reference table.

This table describes the TRANSLATE service return codes:

RETURN	DIAGNOSTIC CODE	DESCRIPTION
CPTIRCOK		Successful
CPTEVERS		Control block version number is not supported
CPTETOKN		Specified token is not valid
CPTEBUFF		Buffer address and/or length is invalid
CPTECHAR		Translation table character set is invalid
CPTEMODE		Translate mode specification is invalid.
CPTEFRMT		Format or specification error
CPTENAPI	Yes	Transport provider API is not available
CPTABEND		Abnormal exception occurred
CPTEOTHR		An undefined exception occurred

200801-024040-200100

Usage Information

The TRANSLATE service translates data between EBCDIC and ASCII. The requirement for translation is application dependent.

The version number (axl_vers) indicates the CPT release level in which this user application program is written. This required field must be set to AXL_VERSN and is validated by the TRANSLATE service before processing the request.

The function code (axl_func) indicates the CPT callable service ID. The field is not initialized by a user application program and has little value to the application except for dump analysis. The function code identifies and maps an argument list with the error or trace log and dump analysis.

The token (axl_token) indicates the connection associated with this translation request. This field is required; however, no transport provider requests are issued. The token is used for internal logging support requirements. This required field is validated by the TRANSLATE service before processing the request.

The axl_xmode field specifies the character set mode. This field sets single, double, or mixed character set translation. Currently, only single-byte character set translation, which is the default, is supported.

The axl_xtype field specifies the translation direction. This required field indicates EBCDIC to ASCII, or ASCII to EBCDIC. Additionally, characters can be transacted into the corresponding uppercase values.

20 024040-200100

Example:

This example establishes a connection and sends data to the remote host. The data is translated from EBCDIC to ASCII before the CPT SEND request is issued. The token is loaded from the ACM and used by all of the following CPT service requests.The axl_rtncd field is checked to determine TRANSLATE service request completion status.

```
#include  <t09ksacm.h>
#include  <t09ksaxl.h>
#include  <t09ksrcs.h>

void main()
{
    struct adt_stru
        cpt_adt = {2, 0, 0, NULL, NULL, 0, 0, 0, 0};

    strut axl_stru
        cpt_axl = {2, 0, 0, NULL, NULL, 0, 0, 0, 0};

char message[256];

    /*
    *  CPT Connection Management service request
    */

    cpt_adt.adt_token = cpt_acm.acm_token;

    /*
    *  Data Processing Routine
    */

    while (data)
    {
        cpt_axl.axl_saddr = &message;
        cpt_axl.axl_sleng = msglen;
        cpt_axl.axl_xtype = AXLXTYPE_ETOA;
        t09fxlat (&cpt_axl);
        if (cpt_axl.axl_rtncd != 0)
        {
            /*
            * process CPT TRANSLATE service error and terminate transaction;
            */
        }

        cpt_adt.adt_buffa = &msgarea;
        cpt_adt.adt_buffl = msglen;
        t09fsend (&cpt_adt);
        if (cpt_adt.adt_rtncd != 0)
        {
            /*
            * process CPT SEND service error and terminate transaction;
            */
        }
    }

    /*
    *  CPT Release Connection
    */

    /*
    *  Terminate Transaction
    */

    End:
    EXEC CICS RETURN;
}
```

200801-024040-200100

Example:

This example establishes a connection and sends data to the remote host. The data is read from the network connection and translated from ASCII to EBCDIC. The token is loaded from the ACM and used by all of the following CPT service requests. The `axl_rtncd` field is checked to determine `TRANSLATE` service request completion status.

```
#include  <t09ksaxl.h>
#include  <t09ksadt.h>
#include  <t09ksacm.h>
#include  <t09ksrcs.h>

void main()
{
    struct adt_stru
        cpt_adt = {2, 0, 0, NULL, NULL, 0, 0, 0, 0};

    struct axl_stru
        cpt_axl = {2, 0, 0, NULL, NULL, 0, 0, 0, 0};

    char message[256];

    /*
    *  CPT Connection Management service request
    */

    cpt_adt.adt_token = cpt_acm.acm_token;

    /*
    *  Data Processing Routine
    */

    while (data)
    {
        cpt_adt.adt_buffa = &msgarea;
        cpt_adt.adt_buffl = msglen;
        t09frecv (&cpt_adt);
        if (cpt_adt.adt_rtncd != 0)
        {
            /*
            * process CPT RECEIVE service error and terminate transaction;
            */
        }
        cpt_axl.axl_saddr = cpt_adt.adt_buffa;
        cpt_axl.axl_sleng = cpt_adt.adt_buffl;
        cpt_axl.axl_xtype = AXLXTYPE_ATOE;
        t09fxlat (&cpt_axl);
        if (cpt_axl.axl_rtncd != 0)
        {
            /*
            * process CPT TRANSLATE service error and terminate transaction;
            */
        }
    }

    /*
    *  CPT Release Connection
    */

    /*
    *  Terminate Transaction
    */

    End:
    EXEC CICS RETURN;
}
```

Return Codes

The T09KSRCS header file resolves API service return codes.

```
/*************************************************************/
/*  Exception Codes Posted in Return Code Argument Fields    */
/*************************************************************/
#define CPTIRCOK 0   /* Request completed successfully       */
/*************************************************************/
/*                        WARNINGS                           */
/*************************************************************/
#define CPTWTIMO 1   /* TIMED RECEIVE SERVICE CALL TIMED OUT     */
#define CPTWNEGO 4   /* BUFFER, QUEUE SIZES RESET TO SYSTEM LIMITS*/
#define CPTWBLCK 6   /* RECEIVE WOULD BLOCK - NO DATA AVAILABLE  */
#define CPTWNEOM 8   /* INCOMPLETE DATAGRAM                      */
#define CPTWNSEP 10  /* NO SEPARATOR CHARACTERS FOUND            */
#define CPTWEXCP 15  /* OTHER WARNING                            */
/*************************************************************/
/*            CONTROL BLOCK ARGUMENT ERRORS                  */
/*************************************************************/
#define CPTEVERS 17  /* CONTROL BLOCK VERSION NUMBER NOT SUPPORTED*/
#define CPTECONN 18  /* REQ HOST/SERVICE/PORT CONNECTION NOT FOUND*/
#define CPTEPROT 19  /* SPECIFIED PROTOCOL NOT SUPPORTED      */
#define CPTETOKN 20  /* SPECIFIED DATA TRANSFER TOKEN IS INVALID*/
#define CPTEBUFF 21  /* BUFFER ADDRESS AND/OR LENGTH INVALID */
#define CPTECHAR 22  /* TRANSLATE CHARACTER SET IS INVALID    */
#define CPTEMODE 23  /* TRANSLATE MODE SPECIFICATION IS INVALID*/
#define CPTECOPT 24  /* CLOSE MODE SPECIFICATION IS INVALID   */
#define CPTETABL 25  /* SPECIFIED TRANSLATE TABLE NOT CORRECT*/
#define CPTETRID 26  /* DESIGNATED TRANSACTION ID CANNOT START*/
#define CPTETIME 27  /* RECEIVE TIMEOUT VALUE NOT SPECIFIED      */
#define CPTESEPC 28  /* NUMBER OF SEPARATOR CHARACTERS NOT SPECIFD*/
#define CPTEOPTN 29  /* RECEIVE OPTIONS IN CONFLICT              */
#define CPTEOPRL 30  /* RECEIVE OPTION NOT AVAILABLE (TCP RELEASE)*/
#define CPTEFRMT 31  /* OTHER TPL FORMAT OR SPECIFICATION ERROR*/
/*************************************************************/
/*            LOCAL ENVIRONMENT ERRORS                       */
/*************************************************************/
#define CPTEPBSY 33  /* SELECTED PORT IS BUSY WITH ACTIVE SERVER*/
#define CPTENAPI 34  /* SNS/API/CICS NOT FULLY AVAILABLE, RETRY*/
#define CPTENAVL 35  /* RESTED FACILITY IS NOT AVAILABLE      */
#define CPTEDRAN 36  /* ENVIRONMENT IS BEING DRAINED          */
#define CPTETERM 40  /* ENVIRONMENT IS BEING TERMINATED       */
#define CPTEENVR 47  /* OTHER TPL ENVIRONMENTAL CONDITION     */
/*************************************************************/
/*            CONNECTION EXCEPTIONS                          */
/*************************************************************/
#define CPTERLSE 65  /* ORDERLY RELEASE OF REMOTE CONNECTION REQ*/
#define CPTEDISC 68  /* REMOTE CONNECTION NOT AVAILABLE OR ABORTED*/
#define CPTEPRGE 72  /* REMOTE CONNECTION ENVIRONMENT TERMINATING*/
#define CPTEINTG 79  /* OTHER TPL CONNECTION/DATA INTEGRITY ERROR*/
/*************************************************************/
/*            OTHER EXCEPTIONS                               */
/*************************************************************/
#define CPTEPROC 143 /* PROCEDURAL ERROR                      */
#define CPTABEND 254 /* ABNORMAL TERMINATION                  */
#define CPTEOTHR 255 /* OTHER ERROR                           */
```

200801-024040-200100

PARAMETER	DESCRIPTION
CPTIRCOK	Request Completed Successfully Indicates that the requested service completed successfully.
CPTWTIMO	Timed RECEIVE Service Call Timed Out See the following RECEIVE service options: ADTTYPLL ADTTYPSP ADTTMRCV ASTTMPRT
CPTWNEGO	System Limits Applied to Buffer and Queue Sizes A Connection Management service returned modified API buffering values. The ACMQSEND, ACMMSEND, ACMQRECV or ACMMRECV values have been modified during transport provider connection negotiation. This return code does not generally indicate a serious error, but is a warning.
CPTWNEGO	Buffer, Queue Sizes Reset to System Limits A connection management service returned modified API buffering values. The ACMQSEND, ACMMSEND, ACMQRECV or ACMMRECV values were modified during transport provider connection negotiation. This return code does not generally indicate a serious error, but rather a warning
CPTWBLCK	RECEIVE Would Block - No Data Available No data was available on a TCP connection if the call was to the RECEIVE service, or no datagrams were available at a UDP endpoint if the call was to the RCVFROM service. In both cases, ADTOPCD1 was set to ADTNBLKR.
CPTWNEOM	Incomplete Datagram On a RCVFROM service call, ADTBUFFA for a length of the ADTBUFFL was not large enough to hold the entire datagram. Subsequent RCVFROM calls will be required to retrieve the rest of the datagram.
CPTWNSEP	No Separator Characters Found On a RECEIVE service call with the SEP option specified (ADTTYPSP) no indicated separator sequence was found in the data. Check user data and make sure the ADTTIMEO value was long enough to receive all of the data.
CPTWEXCP	Transport Provider Exceptional Condition A transport provider exceptional error has occurred. Review diagnostic code for additional information.
CPTEVERS	Argument Version Number not Supported Indicates that the version number specified is not supported. The service request is not executed.
CPTECONN	Requested Host/Service Port Connection not Found A connection management service request failed due to an invalid or unresolved host, service name or port number specification. The service request is not executed.
CPTEPROT	Specified Protocol not Supported A connection management service request failed due to an invalid or unsupported protocol specification. The protocol specified was not TCP or UDP. The service request is not executed.

2[illegible]-024040-200100

PARAMETER	DESCRIPTION
CPTETOKN	Specified Token is Invalid The specified token in service argument is invalid. The service request is not executed.
CPTEBUFF	Buffer Address and/or Length Invalid The specified buffer address and/or length is invalid. A data transfer or data translation service request failed verification of buffer address or length. The service request is not executed.
CPTCHAR	Translate Character Set is Invalid The specified translation character set is invalid. The translation service request is not executed.
CPTEMODE	Translate Mode Specification is Invalid The specified translation mode is invalid. The translation service request is not executed.
CPTETRID	Designated Transaction ID cannot be Started The LISTEN service attempted to start a transaction that failed. The LISTEN service request is terminated and the well-known server port is released.
CPTEFRMT	Other Transport Provider Format or Specification Error A transport provider format or specification error occurred. Review diagnostic code for additional information. The service request failed to execute successfully.
CPTEPBSY	Selected Port is Busy with Active Server. The LISTEN service attempted to establish a connection to the local transport provider but the specified port was already allocated. The LISTEN service request failed to execute.
CPTENAPI	API, Transport Provider or CICS not available. The Transport Provider, API Communication Subsystem or CICS is not available. Review diagnostic code for additional information. The service request failed to execute.
CPTEDRAN	Transport Provider or API is Being Drained The Transport Provider or API communication Subsystem is being drained. This indicates that the CPT Programming Interface may be terminating.
CPTETERM	Transport Provider or API is Being Terminated The Transport Provider or API communication Subsystem is being terminated. This indicates that the CPT Programming Interface is terminated. All connections are closed and no communication is available.
CPTEENVR	Other Transport Provider Environment Error A Transport Provider environment error occurred. Review diagnostic code for additional information. The service request failed to execute successfully.
CPTERLSE	Orderly Release of Full Duplex Connection Indicated Indicates that the remote host released its half of the full duplex connection.The connection is still available for communication, but the user application will not receive data for the remote host.

200801-024040-200100

PARAMETER	DESCRIPTION
CPTEDISC	Remote Connection not Available or Aborted Indicates that a connection request failed or an established connection was aborted. The connection request is not established or an established connection is terminated.
CPTEPRGE	CICS Programmer's Toolkit Interface Terminated The CPT Interface has been terminated. The CPT termination transaction has been initiated by either the termination transaction or CICS shutdown PLT entry.
CPTEINTG	Other Transport Provider Integrity Error A Transport Provider integrity error occurred. Review diagnostic code for additional information. The service request failed to execute successfully.
CPTEPROC	Other Transport Provider Procedure Error A Transport Provider procedure error occurred. Review diagnostic code for additional information. The service request failed to execute successfully.
CPTABEND	Abnormal Environmental Error The service request experienced an abnormal termination condition. The service request failed to execute. The CICS dump data set contains additional information related to the abend.
CPTEOTHR	Any Other Currently Undefined Condition An unknown condition was detected. Review the diagnostic code for additional information. The service request failed to execute.

20-024040-200100

CICS Storage Requirements

All CICS storage is shared and allocated from above the 16MB line, if possible. Storage requirements for a TCP connection or a UDP endpoint within the CICS address space are:

- `1080 + ((72 + xxxMSEND) * xxxQSEND) + ((72 + xxxMRECV) * xxxQRECV)`

  where `xxx` is either ACM for TCP connections or ADT for UDP endpoints

Note: For TCP server applications, an additional 1080-byte overhead is required to allocate the listening token/socket. This computation accounts only for the TCP connection that is subsequently established by a listening token/socket or by the storage for a UDP endpoint.

Note: The send buffer queue is allocated only if an application calls the `SEND` or `SENDTO` services. Determining the `xxxMSEND` and `xxxQSEND` values is explained below. The receive buffer queue is allocated only if an application calls the `RECEIVE` or `RCVFRM` services. Determining the `xxxMRECV` and `xxxQRECV` values is explained below.

Note: The receive buffer queue is allocated only if an application calls the `RECEIVE` or `RECVFROM` services. Determining the `xxxMRECV` and `xxxQRECV` values is explained below.

The maximum send and receive data sizes, the maximum send and receive buffer sizes, and the send and receive queues' sizes (maximum number of outstanding requests per endpoint) are determined initially by the `ACPCONFG` macro, `ACFTIB`. Read the ***SNS/TCPaccess Customization Guide***, pages A-70 and A-71, for information. The TCP maximum data size default that can be sent or received is 32K. The maximum data buffer size default is 64K. CPT negotiates with SNS/PCaccess to determine these values:

- Determine final `xxxQSEND` value:

```
If xxxSEND is not specified,
       use the lesser of 4 or the value in DFQSEND
   else
       use the lesser of the value specified in xxxQSEND or
       DFQSEND.
```

- Determine final `xxxQRECV` value:

```
If xxxQRECV is not specified,
       use the lesser of 4 or the value in DFGRECV
   else
       use the lesser of the value specified in xxxQRECV or
       DFQRECV.
```

- Determine final `xxxMSEND` value:

```
If xxxMSEND is not specified,
       use the lesser of 4096 or the value in MXLTSND
   else
       use the lesser of the value specified xxxMSEND or
```

200801-024040-200100

```
        MXLTSND
either of which must be less than the following
computation:
        (the lesser of DFLSEND or 61440) / the final
        xxxQSEND value
    else
        use the quotient from this computation as thefinal
        xxxMSEND value.
```

- **Determine final xxxMRECV value:**

```
If xxxMRECV is not specified,
        use the lesser of 4096 or the value in MXLTRCV
    else
        use the lesser of the value specified in xxxMSEND or
        MXLTRCV
either of which must be less than the following
computation:
(the lesser of DFLRECV or 61440) / the final xxxQRECV value
    else
        use the quotient from this computation as the final
        xxxMRECV value.
```

Chapter 4

COBOL SUBROUTINE CALLS

This chapter describes the COBOL subroutine calls of CICS/API. These are:

- CLOSE
- CONNECT
- GIVE
- LISTEN
- RCVFROM
- RECEIVE
- SEND
- SENDTO
- TAKE
- TRANSLATE

It also describes the return codes of the T09KCRCS copy member, the differences between OS/VS COBOL and COBOL II, and the CICS storage requirements for a TCP connection or a UDP endpoint.

All messages are in **Appendix C – MESSAGES AND CODES** in the ***CICS Programmer's Toolkit Installation and Administration Guide***.

200801-024040-200100

CLOSE

The CLOSE service closes an established connection. Both orderly (or graceful) and abortive termination options are supported. The CLOSE service performs all associated functions required for CPT resource clean-up.

To invoke the CLOSE service, a user application is required to first build an Argument for Close (ACL) and then to issue a call to the CLOSE routine. Valid arguments include the ACL version number, connection token, and termination options. On completion, a return code is set to indicate success or failure of the request.

This table describes the arguments for the CLOSE service:

COPY NAME	SIZE	CREATED BY
T09KCACL	30 (X'1E')	User application

Structure

This is the structure of the CLOSE service in COBOL language:

```
  01  CPT-ACL.
                COPY T09KCACL.
    *
*****************************************************************
*                          CPT-ACL                              *
*        CICS PROGRAMMER'S TOOLKIT CLOSE CONTROL BLOCK          *
*---------------------------------------------------------------*
          05  ACLVERS          PIC  9(4) COMP VALUE 2.
              88 ACLVERSN      VALUE 2.
          05  ACLFUNC          PIC  9(4) COMP VALUE 0.
          05  ACLTOKEN         PIC  9(8) COMP VALUE 0.
          05  FILLER           PIC  9(8) COMP VALUE 0.
          05  FILLER           PIC  9(8) COMP VALUE 0.
          05  ACLRTNCD         PIC  9(8) COMP VALUE 0.
              88 ACL-RCOKAY    VALUE  0 THRU  15.
              88 ACL-ERROR     VALUE 16 THRU 255.
          05  ACLDGNCD         PIC  9(8) COMP VALUE 0.

*---------------------------------------------------------------*
*  ACLOPTNS         TERMINATION OPTIONS                         *
*---------------------------------------------------------------*
          05  ACLOPTNS         PIC  9(8) COMP VALUE 0.
      * ORDERLY RELEASE:
              88 ACLOPT-ORDER  VALUE 0.
      * ABORTIVE TERMINATION:
              88 ACLOPT-ABORT  VALUE 1.
      *
           05  FILLER           PIC  9(4) COMP VALUE 0.
```

The following constants are contained in the T09MAC library member, T09KCCON:

```
01  ACL-CONSTANTS.

*---------------------------------------------------------------*
*  ACLOPTNS          TERMINATION OPTIONS                        *
*---------------------------------------------------------------*
     * ORDERLY RELEASE:
        05 ACLOPTNS-ORDER     PIC  9(8) COMP VALUE 0.
     * ABORTIVE TERMINATION:
        05 ACLOPTNS-ABORT     PIC  9(8) COMP VALUE 1.
```

PARAMETER	DESCRIPTION
ACLVERS	Version number Indicates the CPT version number of the argument list used by the calling program. This required field is set to 2 for this release of CPT. Default: 2
ACLFUNC	Function code Indicates the function or callable service ID requested by the application program. This field is not set by the application, but is initialized by the *Task-Related User Exit* (*TRUE*) interface stub program. Default: None
ACLTOKEN	Connection or endpoint token Specifies a token that represents the connection. A token is obtained from a connection management request. The token is required. Default: None
ACLRTNCD	Return code Indicates the return code set by the CLOSE service. This value indicates the success or failure of the service. Default: 0
ACLDGNCD	Diagnostic code Indicates the diagnostic code set by the service request. This value generally indicates a transport provider return code. Default: 0
ACLOPTNS	Specifies CLOSE processing control options. These are the options supported: ◆ ACLOPT-ORDER – Indicates a graceful termination. This option implements orderly release of the TCP/IP connection. This is the preferred option for terminating a connection, and is used when processing has completed successfully. ◆ ACLOPT-ABORT – Indicates abortive termination. This option implements a disconnect or reset of the TCP/IP connection. This option is generally used after an unrecoverable application error has occurred. **Note:** The notion of orderly or abortive CLOSE for a UDP endpoint is meaningless and the options specified when calling CLOSE for a UDP token are not important. CPT knows if the token is UDP and will close it properly. Default: ACLOPT-ORDER

Completion Information

The CLOSE service completes normally when the connection is terminated and associated resources are released. Graceful termination waits for all pending transport provider asynchronous SEND and RECEIVE requests to complete. Graceful termination also waits for both ends of the full-duplex connection to close. Abortive termination closes the transport provider connection without regard to pending transport provider requests. Abortive termination can cause data loss and should be used only when data integrity is not required.

On normal return to the application program, the general return code in ACLRTNCD is set to zero (CPTIRCOK). The diagnostic code in ACLDGNCD is always zero.

If the CLOSE service completes abnormally, some user data may be lost. The general return code in ACLRTNCD, and the diagnostic code in ACLDGNCD, indicate the nature of the failure. The diagnostic code (ACLDGNCD) may contain a specific code which identifies a particular transport provider error.

200801-024040-200100

Return Codes

The CLOSE service return and diagnostic codes indicate the result of the execution. These values are in the ACLRTNCD and ACLDGNCD within the Argument for Connection Management (ACM). The diagnostic code is optional and indicates the transport provider return code. Read **Appendix C – MESSAGES AND CODES** in the ***CICS Programmer's Toolkit Installation and Administration Guide*** for the return code cross reference table.

This table describes the CLOSE service return codes:

RETURN	DIAGNOSTIC CODE	DESCRIPTION
CPTIRCOK		Successful.
CPTEVERS		Control block version number is not supported.
CPTETOKN		Specified token is not valid.
CPTEPRGE	Yes	CPT Interface is terminating.
CPTENAPI	Yes	Transport provider API is not available.
CPTETERM	Yes	Environment is being terminated.
CPTERLSE	Yes	Release indication.
CPTEDISC	Yes	Disconnect indication.
CPTEINTG	Yes	Transport provider API integrity error.
CPTEENVR	Yes	Transport provider API environment error.
CPTEFRMT	Yes	Transport provider API format error.
CPTEPROC	Yes	Transport provider API procedure error.
CPTABEND		Abnormal exception occurred.
CPTEOTHR	Yes	An undefined exception occurred.

208024040-200100

Usage Information

The CLOSE service terminates an established transport provider endpoint and releases associated resources. Established transport provider connections can be half of a TCP connection, a TCP listening endpoint, or a UDP endpoint, and are represented by a token.

The CLOSE service utilizes the ACL. The CLOSE service requires the application to set the ACL version number and token fields. Optional control information related to termination processing can be specified. The address of the ACL is required to be loaded into register 1 before the CLOSE service.

The version number (ACLVERS) indicates the CPT release level in which this user application program is written. This required field must be set to ACLVERS2 and is validated by the CLOSE service before it processes the request.

The function code (ACLFUNC) indicates the CPT callable service ID. The field is not initialized by a user application program and has little value to the application except for dump analysis. The function code can identify and map an argument list with the error or trace log and dump analysis.

The token (ACLTOKEN) indicates the connection and internal resources to be released. This is a required field and is validated by the CLOSE service before it processes the request.

The ACLOPTCD field specifies CLOSE processing control options and provides a mechanism for event notification on return to the application program. Currently, the options supported are ACLOPT-ORDER and ACLOPT-ABORT; no facility exists for CLOSE event notification, except by way of return code values.

If the option code ACLOPT-ORDER is selected, the ACLOPT-CLOSE service completes all pending transport provider requests. These requests represent previous asynchronous SENDs and/or RECEIVEs that have neither completed nor had their completion checked. This may require the CLOSE service to block the application. This option then performs an orderly release of the TCP/IP connection. This is the preferred mechanism for connection termination.

If the option code ACLOPT-ABORT is selected, the CLOSE service terminates the connection and makes no attempt to preserve data in transit. The remote user receives a disconnect indication.

200801-024040-200100

Example:

This example establishes a connection, processes data, and closes the connection. The token is loaded from the ACM and used by all of the following CPT service requests. Orderly connection termination is requested. The return code is checked to determine CLOSE service completion status.

```
WORKING-STORAGE SECTION.
01     CPT-ACM   COPY T09KCACM.
01     CPT-ADT   COPY T09KCADT.
01     CPT-ACL   COPY T09KCACL.
COPY T09KCRCS.
PROCEDURE DIVISION.

*
* CPT Connection Management initialization and request
*
    .
    .
    .
    MOVE ACMTOKEN TO ADTTOKEN ACLTOKEN.

*
* Application and CPT data transfer (SEND/RECEIVE) processing
*
    PERFORM UNTIL NOT ADT-RCOKAY
    .
    .
    .
      END-PERFORM.

*
* CPT Orderly Connection Release processing
*
    SET ACLOPT-ORDER TO TRUE.
    CALL 'T09FCLOS' USING CPT-ACL.
    IF NOT ACL-RCOKAY
*
*   . Process and log CLOSE service error
*
    END-IF.
*
* Terminate Transaction
*
    EXEC CICS RETURN
    END-EXEC.
```

2C 024040-200100

Example:

This example establishes a connection, processes data, and closes the connection. The token is loaded from the Argument for Connection Management (ACM) and used by all of the following CPT service requests. An error is received processing data and the abortive termination option is selected. The return code is checked to determine CLOSE service completion status.

```
WORKING-STORAGE SECTION.
01     CPT-ACM    COPY T09KCACM.
01     CPT-ADT    COPY T09KCADT.
01     CPT-ACL    COPY T09KCACL.
COPY CTPKCRCS.

PROCEDURE DIVISION.

*
* CPT Connection Management initialization and request
*
    .
    .
    .
    MOVE ACMTOKEN TO ADTTOKEN ACLTOKEN.

*
* Application and CPT data transfer (SEND/RECEIVE) processing
*
    PERFORM UNTIL NOT ADT-RCOKAY
    .
    .
    .
    END-PERFORM.

*
* CPT Abortive Connection Release processing
*
    SET ACLOPT-ABORT TO TRUE.
    CALL 'T09FCLOS' USING CPT-ACL.
    IF NOT ACL-RCOKAY
*

    . Process and log CLOSE service error
*
    END-IF.
*
* Terminate Transaction
*
    EXEC CICS RETURN
    END-EXEC.
```

200801-024040-200100

Example:

This example terminates a single thread server application. A server application can contain two CPT connections; the first for the data transfer connection and the second for the server or listening connection. The tokens are determined from the ACM. Orderly connection termination is requested for both the data transfer and listening connections. The return code is checked to determine CLOSE service completion status.

```
WORKING-STORAGE SECTION.
01     CPT-ACM    COPY T09KCACM.
01     CPT-ADT    COPY T09KCADT.
01     CPT-ACL    COPY T09KCACL.
COPY CTPKCRCS.
PROCEDURE DIVISION.

*
* CPT Connection Management initialization and request
*

    .
    .
    .
    MOVE ACMTOKEN TO ADTTOKEN ACLTOKEN.

*
* Application and CPT data transfer (SEND/RECEIVE) processing
*
    PERFORM UNTIL NOT ADT-RCOKAY
    .
    .
    .
    END-PERFORM.

*
* CPT Orderly Connection Release processing
*
    SET ACLOPT-ORDER TO TRUE.
    CALL 'T09FCLOS' USING CPT-ACL.
    IF NOT ACL-RCOKAY
*
*   . Process and log CLOSE service error
*
    END-IF.
    MOVE ACMTLSTN TO ACLTOKEN.
    SET ACLOPT-ORDER TO TRUE.
    CALL 'T09FCLOS' USING CPT-ACL.
    IF NOT ACL-RCOKAY
*
*    . Process and log CLOSE service error
*
    END-IF.

*
* Terminate Transaction
*
    EXEC CICS RETURN
    END-EXEC.
```

20-024040-200100

CONNECT

The CONNECT service provides a client facility for use by an application program. The CONNECT service establishes a session with the local transport provider, then actively connects to a server. When connection is established with a server, the CONNECT service returns control to the calling program. Information related to the connection is updated and returned within the ACM.

To invoke the CONNECT service, a user application is required to first build an ACM and then to issue a call to the CONNECT routine. The minimum information required by this service is version number, server host address, and well-known port. Optional information related to data transfer buffering, CPT statistics and tracing, and subtask initialization can be specified.

This table describes the CONNECT service arguments:

COPY NAME	SIZE	CREATED BY
T09KCACM	676 (X'2A4')	User application

Structure

This is the structure of the CONNECT service in COBOL language:

```
 01  CPT-ACM.
                 COPY T09KCACM.

*****************************************************************
*                          CPT-ACM                              *
*     CICS PROGRAMMER'S TOOLKIT CONNECTION MANAGEMENT BLOCK     *
*---------------------------------------------------------------*
          05  ACMVERS          PIC   9(4) COMP VALUE 2.
              88  ACMVERSN     VALUE 2.
          05  ACMFUNC          PIC   9(4) COMP VALUE 0.
          05  ACMTOKEN         PIC   9(8) COMP VALUE 0.
          05  FILLER           PIC   9(8) COMP VALUE 0.
          05  FILLER           PIC   9(8) COMP VALUE 0.
          05  ACMRTNCD         PIC   9(8) COMP VALUE 0.
              88 ACM-RCOKAY    VALUE  0.
              88 ACM-WARNING   VALUE  1 THRU  15.
              88 ACM-ERROR     VALUE 16 THRU 255.
          05  ACMDGNCD         PIC   9(8) COMP VALUE 0.
          05  ACMSTATS         PIC   9(8) COMP VALUE 0.
          05  ACMTRACE         PIC   9(8) COMP VALUE 0.
          05  ACMQSEND         PIC   9(8) COMP VALUE 0.
          05  ACMMSEND         PIC   9(8) COMP VALUE 0.
          05  ACMQRECV         PIC   9(8) COMP VALUE 0.
          05  ACMMRECV         PIC   9(8) COMP VALUE 0.
          05  ACMTLSTN         PIC   9(8) COMP VALUE 0.
          05  ACMUCNTX         PIC   9(8) COMP VALUE 0.
          05  ACMTRNID         PIC   X(4) VALUE LOW-VALUES.
          05  FILLER           PIC   X(1) VALUE LOW-VALUES.
          05  FILLER           PIC   X(1) VALUE LOW-VALUES.
          05  FILLER           PIC   9(4) COMP VALUE 0.
          05  ACMLPORT         PIC   9(4) COMP VALUE 0.
```

200801-024040-200100

```
05  ACMRPORT          PIC  9(4) COMP VALUE 0.
05  ACMSRVCE          PIC X(36) VALUE SPACES.
05  FILLER            PIC  X(1) VALUE LOW-VALUES.
05  FILLER            PIC  X(1) VALUE LOW-VALUES.
05  ACMOPTNS          PIC  9(4) COMP VALUE 0.
05  ACMLADDR          PIC  9(9) COMP VALUE 0.
05  ACMRADDR          PIC  9(9) COMP VALUE 0.
05  ACMLNAME          PIC X(255) VALUE SPACES.
05  FILLER            PIC  X(1) VALUE LOW-VALUES.
05  ACMRNAME          PIC X(255) VALUE SPACES.
05  FILLER            PIC  X(1) VALUE LOW-VALUES.
05  FILLER            PIC  9(4) COMP VALUE 0.
05  FILLER            PIC  9(4) COMP VALUE 0.
05  ACMTIMEO          PIC  9(8) COMP VALUE 0.
05  FILLER            PIC  9(8) COMP VALUE 0.
```

The following constants are contained in the T09MAC library member, T09KCCON:

```
   01  ACM-CONSTANTS.

*-----------------------------------------------------------------*
*   ACMSTATS          STATISTICS LOG REQUESTS                     *
*-----------------------------------------------------------------*
     * CONNECTION STATISTICS:
        05  ACMSTATS-CONN    PIC  9(4) COMP VALUE 1.
     * TERMINATION STATISTICS:
        05  ACMSTATS-TERM    PIC  9(4) COMP VALUE 2.

*-----------------------------------------------------------------*
*   ACMTRACE            TRACE LOG REQUESTS                        *
*-----------------------------------------------------------------*
     * TRACE ENTRY POINTS:
        05  ACMTRACE-NTRY    PIC  9(4) COMP VALUE 1.
     * TRACE ARGUMENTS:
        05  ACMTRACE-ARGS    PIC  9(4) COMP VALUE 2.
     * TRACE TRECV:
        05  ACMTRACE-RECV    PIC  9(4) COMP VALUE 4.
     * TRACE TSEND:
        05  ACMTRACE-SEND    PIC  9(4) COMP VALUE 8.
     * TRACE TERMINATION:
        05  ACMTRACE-TERM    PIC  9(4) COMP VALUE 16.
     * TRACE TAKE:
        05  ACMTRACE-PASS    PIC  9(4) COMP VALUE 32.
     * TRACE CLOSE:
        05  ACMTRACE-CLSE    PIC  9(4) COMP VALUE 64.
     * TRACE TPLERRORS:
        05  ACMTRACE-TERR    PIC  9(4) COMP VALUE 128.
     * TRACE TOKENS:
        05  ACMTRACE-TOKN    PIC  9(4) COMP VALUE 256.
     * TRACE TPL:
        05  ACMTRACE-TPL     PIC  9(4) COMP VALUE 512.
     * TRACE RELEASE:
        05  ACMTRACE-RLSE    PIC  9(4) COMP VALUE 1024.
     * TRACE STORAGE MANAGEMENT:
        05  ACMTRACE-STOR    PIC  9(4) COMP VALUE 2048.
     * TRACE CLIENT DATA:
        05  ACMTRACE-CLTD    PIC  9(4) COMP VALUE 4096.
```

20 *024040-200100

```
*-------------------------------------------------------------*
*   ACMOPTNS                PROCESSING OPTIONS                *
*-------------------------------------------------------------*
     * ISSUE SYNCPOINT FROM LISTEN
        05  ACMOPTNS-SYNC     PIC  9(4) COMP VALUE   1.
     * CLIENT-DATA LISTENER OPTION
        05  ACMOPTNS-LTRAN    PIC  9(4) COMP VALUE   2.
     * NO DNR NAME RESOLUTION
        05  ACMOPTNS-NODNR    PIC  9(4) COMP VALUE   4.
```

PARAMETER	DESCRIPTION
ACMVERS	Version Indicates the version number of the CPT argument list used by the calling program. This required field is set to 2 for this release of CPT. Default: 2
ACMFUNC	Function code Indicates the function or callable service ID requested by the application program. This field is not set by the application, but rather is initialized by the TRUE interface stub program. Default: None (generated by service stub)
ACMTOKEN	TCP connection token Specifies the token is created and returned by the CONNECT service. This token is used for all subsequent service calls for the client connection. Default: 0 (token returned)
ACMRTNCD	Return code Indicates the return code set by the CONNECT service. This value indicates the success or failure of the service. Default: 0
ACMDGNCD	Diagnostic code Indicates the diagnostic code received by the CONNECT service for a transport provider request. A detailed explanation of this value can be found in the transport provider's ***API Programmer's Reference Guide***. Default: 0
ACMSTATS	Specifies statistics logging options for the application program. The facility can be used for debugging and tuning during development. ◆ ACMSTATS-CONN Specifies that a message(s) be generated on establishing either a listen service or a data transfer connection. These messages are generated by the CPT LISTEN and CONNECT services. The message numbers associated with this option are CPT802I and CPT803I. ◆ ACMSTATS-TERM Specifies that a message(s) is to be generated on termination of an established connection. These messages are generated by the CPT CLOSE service. The message numbers associated with this option are CPT807I, CPT808I, and CPT809I. Default: 0 (no statistics logging)

200801-024040-200100

PARAMETER	DESCRIPTION
ACMTRACE	Specifies trace logging options for the application program. The facility can be used for debugging during development. ◆ ACMTRACE-NTRY – Specifies that a message be generated on entry to a CPT service routine. The message is generated by all CPT service routines. The message number associated with this option is CPT901I. ◆ ACMTRACE-ARGS – Specifies that a hex dump of the caller's arguments be generated on entry and exit of a CPT service routine. Messages are generated by the corresponding service routine. Message numbers associated with this option are CPT902I, CPT903I, CPT904I, CPT911I, CPT912, CPT921I, CPT922I, CPT924I, CPT925I, CPT930I, and CPT932I. ◆ ACMTRACE-RECV – Specifies that a hex dump of the transport provider's input (RECEIVE) data be logged. Messages are generated by the CPT RECV service. The message number associated with this option is CPT913I. ◆ ACMTRACE-SEND – Specifies that a hex dump of the transport provider's output (SEND) data be logged. Messages are generated by the CPT SEND service. The message number associated with this option is CPT914I. ◆ ACMTRACE-TERM – Specifies that a message be generated on termination of an CPT service routine. Messages are generated by all CPT service routines. The message number associated with this option is CPT908I. ◆ ACMTRACE-PASS – Specifies that a hex dump of resources related to a passed connection be logged. Messages are generated by the CPT GIVE and TAKE services. The message numbers associated with this option are CPT919I and CPT920I. ◆ ACMTRACE-CLSE – Specifies that a hex dump of resources related to a CLOSE processing of a connection be logged. Messages are generated by the CPT CLOSE service. The message number associated with this option is CPT906I ◆ ACMTRACE-TERR – Specifies that a hex dump of a transport provider API parameter list that fails successful completion be logged. The message number associated with this option is CPT400I. ◆ ACMTRACE-TOKN – Specifies that a hex dump of the connection token be logged. Messages are generated by all services routines on entry. The message number associated with this option is CPT909I. ◆ ACMTRACE-TPL – Specifies that a hex dump of the transport provider API parameter list be logged. The message numbers associated with this option are CPT917I and CTP935I. ◆ ACMTRACE-RLSE – Specifies that a message indicating a transport provider release indication be logged. Messages are generated by various CPT service routines. The message number associated with this option is CPT414I. ◆ ACMTRACE-STOR – Specifies that a hex dump of storage management argument be logged. Messages are generated by various CPT service routines. The message numbers associated with this option are CPT928I and CPT929I. ◆ ACMTRACE-CLTD – Trace transient data writes from the LISTEN service (used with the ACMOPTNS-LTRAN client-data option). The message number associated with this option is CPT918I. Default: 0 (no trace logging)
ACMQSEND	API send queue size Specifies the maximum number of uncompleted SEND requests that can be queued by the application to the transport provider (API). This value lets applications control output processing and can affect throughput rates. The value is negotiated with and can be modified by the transport provider. The total allocation for output processing is the product of the SEND queue and buffer size values and cannot exceed 61K. Default: 4

20 024040-200100

PARAMETER	DESCRIPTION
ACMMSEND	API send buffer size Specifies the maximum number of user data bytes that can be transferred by the application in a single SEND request to the transport provider (API). This value lets applications control output processing and can affect throughput rates. The value is negotiated with and can be modified by the transport provider. The total allocation for output processing is the product of the SEND queue and buffer size values and cannot exceed 61K. Default: 4096
ACMQRECV	API receive queue size Specifies the maximum number of uncompleted RECEIVE requests that can be queued by the application to the transport provider (API). This value lets applications control input processing and can affect throughput rates. The value is negotiated with and can be modified by the transport provider. The total allocation for input processing is the product of the RECEIVE queue and buffer size values and cannot exceed 61K. Default: 4
ACMMRECV	API receive buffer size Specifies the maximum number of user data bytes that can be transferred by the application, in a single RECEIVE request, to the transport provider (API). This value lets applications control input processing and can affect throughput rates. The value is negotiated with and can be modified by the transport provider. The total allocation for input processing is the product of the RECEIVE queue and buffer size values and cannot exceed 61K. Default: 1024
ACMTLSTN	Listen service token This field is not used by the CONNECT service. The value in this field is not validated nor is it modified. Default: None
ACMUCNTX	One word of user context Specifies one arbitrary word of user context to be associated with the connection. The information provided is not interpreted by CPT, and is merely saved with other connection information. Default: 0 (no user context)
ACMTRNID	Listen start transaction ID This field is not used by the CONNECT service. The value in this field is not validated nor is it modified. Default: None
ACMLPORT	Listen well-known service port This value represents the TCP port on the local host that was assigned to the client application by TCP. It is returned to the caller of the CONNECT service. This field is an unsigned positive integer with a maximum value of 65,534. Default: None
ACMRPORT	Remote well-known service port Indicates the remote transport layer address or port. This value represents the TCP port on the remote host to which the client application is trying to connect. It must be filled in by the calling client application unless the ACMSRVCE field is specified. If ACMSRVCE is specified, ACMRPORT will be filled in with the resolved remote port number before returning to the caller of the CONNECT service. This field is an unsigned positive integer with a maximum value of 65,534. Default: none
ACMSRVCE	Transport layer service name Indicates the local transport layer service name. This string is mapped into a transport layer address or port, through internal Domain Name Resolution (DNR). The resolved value will be the well-known port to which the CONNECT service attempts to establish a connection. This field has a maximum length of 36 bytes. This field is optional and is not modified by the CONNECT service. Default: None

200801-024040-200100

PARAMETER	DESCRIPTION
ACMOPTNS	Specifies TCP connection initialization options. ◆ ACMOPTNS-NODNR – DNR Suppression option. Skips internal DNR calls to resolve and return the remote IP address into an IP name in the ACMRNAME field. If an application is designed such that TCP connection establishment and release happens frequently, this option can save processing time. ◆ ACMOPTNS_LTRAN – Client-Data Listener option. Specifies that the Listen call will receive the input datastream to determine the transaction ID to be started. This option is for the LISTEN service and is not validated or modified by the CONNECT service. ◆ ACMOPTNS-SYNC – Listen Syncpoint option. This option is for the LISTEN service and is not validated or modified by the CONNECT service. Default: None
ACMLADDR	Local IP host address Indicates the local host internet address. This field is an unsigned four-byte integer value. The local host internet address is updated on establishment of a server connection, and will be returned to the caller. Default: None
ACMRADDR	Remote IP host address Indicates the remote host internet address. Either this field or the remote host name (ACMRNAME) field must be specified. This field is an unsigned four-byte integer value. The remote host internet address is updated when a server connection is established, and is returned to the caller. Default: None
ACMLNAME	Local IP host name Indicates the local host internet name. This field is a 255-byte character string that is padded with blanks. The local host internet name is updated when a client connection is established, and is returned to the caller. Default: None
ACMRNAME	Remote IP host name Indicates the remote host internet name. Either this value or the remote IP address (ACMRADDR) field must be specified. This field is a 255-byte character string that is padded with blanks. The remote host internet name is updated when a server connection is established, and is returned to the caller. Default: None
ACMTIMEO	Client-Data Listener timeout value. This field is optionally used by the LISTEN service and is not validated or modified by the CONNECT service. Default: 1 second

2102404-200100

Network Considerations

The ACM is a common data structure used for both client and server connection initialization. There are common and unique values specified for a particular service request.

This table describes network considerations for COBOL API.

NAME	SERVER CONDITIONS FOR LISTEN	CLIENT CONDITIONS FOR CONNECT
ACMTLSTN	Listen token returned to user application.	
ACMTRNID	Listen START transaction ID.	
ACMLPORT	Local server or listening transport provider well-known port selected by user application.	Local assigned transport provider port returned to user application.
ACMRPORT	Remote client transport provider port returned to user application.	Remote server transport provider well-known port selected by user application.
ACMSRVCE	Local server transport provider service name selected by user application.	Remote server transport provider service name selected by user application.
ACMRADDR	Remote IP host address returned to user application.	Remote IP host address selected or returned to user application.
ACMLNAME	Local IP host name returned to user application.	Remote IP host name returned to user application.
ACMRNAME	Remote IP host name returned to user application.	Remote IP host name selected or returned to user application.
ACMTIMEO	Client-Data Listener timeout value.	

200801-024040-200100

Completion Information

The CONNECT service completes normally when a connection with a server is established. The CONNECT service initializes the client environment with the transport provider (API) and actively contacts a server and updates connection information within the ACM. Establishing a client connection is represented by storage and is referred to as the token. When a connection is successfully established the ACM is updated with information related to the connection.

The ACM is updated with information related to the new connection. The local and remote port, IP address, and host names are resolved. Negotiated transport provider SEND and RECEIVE buffering values are returned. The ACM return code (ACMRTNCD) should be checked to determine the success or failure of the CONNECT service. A zero (0) return code indicates a successful connection.

The return and diagnostic codes should be interpreted by the application to determine the reason for failure. Errors indicating CPT, the transport provider

(API), or CICS termination are minor. Errors should be interrogated for level of severity.

Return Codes

The CONNECT service return and diagnostic codes indicate the result of the execution. These values are in the ACMRTNCD and ACMDGNCD within the ACM. The diagnostic code generally indicates the transport provider return code. Read **Appendix C – MESSAGES AND CODES** in the ***CICS Programmer's Toolkit Installation and Administration Guide*** for the return code cross reference table.

This table describes the CONNECT service return codes:

RETURN	DIAGNOSTIC CODE	DESCRIPTION
CPTIRCOK		Successful.
CPTEVERS		Control block version number is not supported.
CPTECONN	Yes	Requested host/service/port is not found.
CPTEPROT		Specified protocol is not supported.
CPTENAPI	Yes	Transport provider API is not available.
CPTETERM	Yes	Environment is being terminated.
CPTERLSE	Yes	Release indication.
CPTEDISC	Yes	Disconnect indication.
CPTEPRGE	Yes	CPT Interface terminating.
CPTEINTG	Yes	Transport provider API integrity error.
CPTEENVR	Yes	Transport provider API environment error.
CPTEFRMT	Yes	Transport provider API format error.
CPTEPROC	Yes	Transport provider API procedure error.
CPTABEND		Abnormal exception occurred.
CPTEOTHR	Yes	An undefined exception occurred.

024040-200100

Usage Information

The CONNECT service lets user-written application programs implement TCP/IP client facilities. The CONNECT service generalized parameter list (ACM) describes the application's communications requirements as well as information related to established connections. On completion, the ACM contains fields initialized by both a user application and by the CONNECT service.

There are required and optional fields initialized by a user or calling application. The ACM version number is required. The server must be identified by the calling program. The server is specified by selecting the remote IP address (ACMRADDR) or host name (ACMRNAME) fields, and the remote port (ACMRPORT) or service name (ACMSRVCE). The selection of port or service name defines the server well-known port address. Optional fields control data transfer buffering, statistics, tracing, and subtask initialization.

On completion of the CONNECT service, the ACM contains information related to the established connection. A token that identifies the connection is returned in the ACM, and must be used in all subsequent requests that refer to the connection. The user application program should make no assumptions regarding the format of a token, other than that it is an unsigned, full word value. Information related to the negotiated buffer values, host names, host addresses, and transport provider addresses are returned in the ACM.

The version number (ACMVERS) indicates the CPT release level in which this user application program is written. This required field must be set to ACMVERS2 and is validated by the CONNECT service before processing the request.

The function code (ACMFUNC) indicates the CPT callable service ID. The field is initialized by the CPT service stub program and has little value to the application except for dump analysis. The function code can identify and map an argument with the error or trace logs, and dump analysis.

The remote IP address (ACMRADDR) or remote host name (ACMRNAME) is required. These fields identify the host to which the CONNECT service initiates a connection request. The IP address has precedence over host name. This implies that the host name field is only used if an IP address is not specified.

The transport provider port number (ACMRPORT) or service name (ACMSRVCE) is required. These fields identify the well-known port to which the CONNECT service will initiate a connection request. The port number has precedence over service name. This implies that the service name field is only used if a port number is not specified.

User application programs have the ability to control CPT and transport provider data transfer buffering. The ACMQSEND, ACMMSEND, ACMQRECV, and ACMMRECV specify the number and size of buffers allocated. The SEND and RECEIVE buffers are allocated on initial entry into either the SEND or RECEIVE service. The corresponding values used by the SEND and RECEIVE services are independent of each other. The SEND service multiplies the queue and buffer values to determine output storage requirements. The RECEIVE service performs a similar function to determine input storage requirements. The product of the queue and buffer values cannot exceed 61,440. CPT requires some additional storage to manage these buffers. This extra storage is included in the allocation.

200801-024040-200100

The SEND service uses the ACMQSEND value; the RECEIVE service uses the ACMQRECV value. These values indicate the maximum number of uncompleted SEND or RECEIVE requests that can be queued by the application to the transport provider. The CPT data transfer services schedule asynchronous transport provider or API requests on the caller's behalf. These API requests must be completed before they can be used again. If the queue values are too small, the CPT data transfer service may block the caller's request and schedule a WAIT command within the service routines. If the queue values are too large, the user application may be wasting storage.

The SEND service uses the ACMMSEND value; the RECEIVE service uses the ACMMRECV value. These values indicate the maximum number of user data bytes that can be transferred by the application in a single SEND or RECEIVE request to the transport provider. The user application is not limited to these values within the data transfer services. However, it is important to note that multiple transport provider or API requests are issued to complete the caller's request. Information on queue and buffer size are in the sections **RECEIVE** and **SEND**.

Initially, the tuning of data transfer storage may not be a concern; however, the ability to control storage allocation can prove beneficial to the application or CICS region. Additionally, queue size can increase data transfer throughput. Consider enabling the statistics option to gather CPT statistical information, which can be used to set the SEND or RECEIVE queue and buffer size values.

The CONNECT service can modify data transfer buffer allocation values. These values are negotiated with the transport provider and, depending on the site configuration, can be reduced. Any application dependent on these values should check them on return. These values are generally not modified when giving reasonable numbers. However, it is advisable to check with the site administrator for the maximum values of the API transport services.

A number of arguments are not set by the calling application, but are returned to the caller. These values represent information related to the client connection and can be used by the application. The local port, host name, and IP address are returned as well as the client's corresponding values.

2024040-200100

Example:

This example establishes a client connection, processes data, and closes the connection. The connection is established to LOCALHOST and the server well-known port is 1234. The token is loaded from the ACM and used by all of the following CPT service requests. The return code is checked to determine CONNECT service completion status.

```
WORKING-STORAGE SECTION.

01  CPT-ACM    COPY T09KCACM.
01  CPT-ADT    COPY T09KCADT.
01  CPT-ACL    COPY T09KCACL.
01  CPT-RCS    COPY T09KCRCS.

PROCEDURE DIVISION.

*
* CPT CONNECT Connection Management initialization and request
*
    MOVE 1234 TO ACMPORT
    MOVE 'LOCALHOST' TO ACMRNAME.
    CALL 'T09FCONN' USING CPT-ACM.
    IF NOT ACM-RCOKAY
*
*   . Process and log CONNECT service error
*   . Terminate Transaction
*
    END-IF.

    MOVE ACMTOKEN TO ADTTOKEN ACLTOKEN.
*
* Application and CPT data transfer (SEND/RECEIVE) processing
*
    PERFORM UNTIL NOT ADT-RCOKAY
    .
    .
    .
    END-PERFORM.

*
* CPT Connection Release processing
*
    .
    .
    .

*
* Terminate Transaction
*
    EXEC CICS RETURN
      END-EXEC.
```

200801-024040-200100

Example:

This example establishes a client connection, processes data, and closes the connection. The connection is established to LOCALHOST and the port is mapped into service name ECHO. The token is loaded from the ACM and used by all of the following CPT service requests. The return code is checked to determine CONNECT service completion status.

```
WORKING-STORAGE SECTION.
01 CPT-ACM    COPY T09KCACM.
01 CPT-ADT    COPY T09KCADT.
01 CPT-ACL    COPY T09KCACL.
01 CPT-RCS    COPY T09KCRCS.

PROCEDURE DIVISION.
*
* CPT CONNECT Connection Management initialization and request
*
     MOVE 'ECHO' TO ACMSRVCE.
     MOVE 'LOCALHOST' TO ACMRNAME.
     CALL 'T09FCONN' USING CPT-ACM.
     IF NOT ACM-RCOKAY
*
*    . Process and log CONNECT service error
*    . Terminate Transaction
*
    END-IF.

    MOVE ACMTOKEN TO ADTTOKEN ACLTOKEN.
*
* Application and CPT data transfer (SEND/RECEIVE) processing
*
    PERFORM UNTIL NOT ADT-RCOKAY
    .
    .
    .
    END-PERFORM.

*
* CPT Connection Release processing
*
    .
    .
    .

*
* Terminate Transaction
*
    EXEC CICS RETURN
    END-EXEC.
```

20-024040-200100

GIVE

The GIVE service releases ownership of a connection and associated internal CPT resources. The GIVE service is optional and does not affect an active connection, nor does it issue any transport provider requests. This service affects CPT TRUE management routines, scheduled on the user's behalf during task termination.

To invoke the GIVE service, a user application is required to first build an Argument for Facility Management (AFM) and then to issue a call to the GIVE routine. The only valid and required arguments are the AFM version number and the connection token. On completion, a return code indicates the success or failure of the request.

This table describes the GIVE service arguments:

COPY NAME	SIZE	CREATED BY
T09KCAFM	34 (X'22')	User application or the LISTEN service

Structure

This is the structure of the GIVE service in COBOL language:

```
   01  CPT-AFM.
               COPY T09KCAFM.

*****************************************************************
*                          CPT-AFM                              *
*  CICS PROGRAMMER'S TOOLKIT FACILITY MANAGEMENT CONTROL BLOCK*
*---------------------------------------------------------------*
         05  AFMVERS          PIC   9(4) COMP VALUE 2.
             88  AFMVERSN     VALUE 2.
         05  AFMFUNC          PIC   9(4) COMP VALUE 0.
         05  AFMTOKEN         PIC   9(8) COMP VALUE 0.
         05  FILLER           PIC   9(8) COMP VALUE 0.
         05  FILLER           PIC   9(8) COMP VALUE 0.
         05  AFMRTNCD         PIC   9(8) COMP VALUE 0.
             88  AFM-RCOKAY   VALUE  0.
             88  AFM-WARNING  VALUE  1 THRU 15.
             88  AFM-ERROR    VALUE 16 THRU 255.
         05  AFMDGNCD         PIC   9(8) COMP VALUE 0.
         05  FILLER           PIC   9(8) COMP VALUE 0.
         05  FILLER           PIC   9(4) COMP VALUE 0.
```

PARAMETER	DESCRIPTION
AFMVERS	Version Indicates the CPT version number of the argument list used by the calling program. This required field is set to 2 for this release of CPT. Default: 2

PARAMETER	DESCRIPTION
AFMFUNC	Function code Indicates the function or callable service ID requested by the application program. This field is not set by the application, but is initialized by the TRUE interface stub program. Default: None
AFMTOKEN	Connection or endpoint token Specifies a token that represents a TCP connection, a TCP listening endpoint, or a UDP endpoint. A token is created by the TCP connection initiation routines or by the UDP data transfer and endpoint creation routines. The token is required. Default: 0
AFMRTNCD	Return code Indicates the return code set by the GIVE service. This value is returned and indicates the success or failure of the service. Default: 0
AFMDGNCD	Diagnostic code Indicates the diagnostic code received by the GIVE service for a transport provider request and is not set by the GIVE service. The GIVE service does not issue transport provider requests; hence, it never sets the diagnostic code. Default: 0

Completion Information

The GIVE service completes normally when all resources associated with this connection are processed.

On normal return to the application program, the general return code in AFMRTNCD is set to zero (CPTIRCOK). The diagnostic code in AFMDGNCD is always zero.

If the GIVE service completes abnormally, some resources associated with this connection cannot be successfully transferred from one task to another. The general return code in AFMRTNCD and the diagnostic code in AFMDGNCD indicate the nature of the failure. The diagnostic code (AFMDGNCD) is not used by the GIVE service and no information is returned.

2024040-200100

Return Codes

The GIVE service return code indicates the result of the execution. This value is in the ACMRTNCD within the AFM. Read **Appendix C – MESSAGES AND CODES** in the ***CICS Programmer's Toolkit Installation and Administration Guide*** for the return code cross reference table.

This table describes the GIVE service return codes:

RETURN	DIAGNOSTIC CODE	DESCRIPTION
CPTIRCOK		Successful
CPTEVERS		Control block version number is not supported
CPTETOKN		Specified token is not valid
CPTENAPI		Transport provider API is not available
CPTABEND		Abnormal exception occurred
CPTEOTHR		An undefined exception occurred

Usage Information

The GIVE service releases ownership of a connection. This service is non-blocking and does not affect any pending transport provider data transfer requests. Dissociating resources from a task lets the CPT properly manage resources during task termination. This ability to GIVE and TAKE ownership of connections offers the user a range of programming options, while still providing CPT with resource management capabilities.

The GIVE service requires the application to set the AFM version number and token fields. No other AFM fields are referenced.

Note: When a connection is established there are internal CPT resources associated with that connection. CPT is responsible for proper clean-up of those resources on task or transaction termination. These resources include storage allocated by CPT, the API, and the transport provider storage.

A server application is a good example of how the GIVE service benefits a user application. A listening task issues the GIVE service and starts a new transaction to handle data transfer. The data transfer transaction then TAKEs the connection. This sequence prevents a connection from being closed (implicitly by the CPT task termination exit) if the server application terminates. However, if the data transfer transaction is terminated without issuing an explicit close (CPT CLOSE service), an implicit close is scheduled, and resource management is handled by the CPT task termination exit.

The version number (AFMVERS) indicates the CPT release level in which this user application program is written. This required field must be set to AFMVERS2 and is validated by the GIVE service before processing the request.

200801-024040-200100

The function code (AFMFUNC) indicates the CPT callable service ID. The field is not initialized by a user application program and has little value to the application except for dump analysis. The function code identifies and maps an argument list with the error or trace log and dump analysis.

The token (AFMTOKEN) indicates the connection and internal resources that are to be processed by the GIVE service. This is a required field and is validated by the GIVE service before processing the request.

The AFMOPTCD field specifies GIVE service processing control options, and provides a mechanism for event notification on return to the application program. Currently, this field is not used by application programs.

Example:

This example establishes a server data transfer connection, issues the GIVE service, and starts a data processing transaction. This example loops for new client connections. The token is loaded from the ACM and used by the GIVE service. The return code is checked to determine GIVE service completion status.

```
WORKING-STORAGE SECTION.
01  CPT-ACM    COPY T09KCACM.
01  CPT-AFM    COPY T09KCAFM.
01  CPT-ACL    COPY T09KCACL.
01  CPT-RCS    COPY T09KCRCS.

PROCEDURE DIVISION.
    PERFORM UNTIL NOT-LOOP
*
*       CPT Connection Management initialization and request
*
    .
    .
    .
        MOVE ACMTOKEN TO AFMTOKEN.

        CALL 'T09FGIVE' USING CPT-AFM.
        IF NOT AFM-RCOKAY
*           Process and log GIVE service error
            SET NOT-LOOP TO TRUE
        ELSE
            EXEC CICS START TRANSID(trans id) FROM(CPT-ACM) LENGTH()
            END-EXEC.
        END-IF.

    END-PERFORM.
*
* CPT Connection Release processing
*
    .
    .
    .
*
* Terminate Transaction
*
    EXEC CICS RETURN
    END-EXEC.
```

LISTEN

This service provides a server facility for use by an application program. The LISTEN service establishes a session with the local transport provider, passively listens for connection requests, then accepts new connections. When connection with a client is established, the LISTEN service either returns control to the calling program or starts a defined transaction. Information related to the connection is updated and returned within the ACM.

To invoke the LISTEN service, a user application is required to first build an ACM, then to issue a call to the LISTEN routine. The minimum information required by this service is the version number and either the local transport provider port or the service name. Optional information related to data transfer buffering, CPT statistics and tracing, and subtask initialization can be specified. Completion of a LISTEN service depends on options selected within the ACM.

This table describes the LISTEN service arguments:

COPY NAME	SIZE	CREATED BY
T09KCACM	676 (X'2A4')	User application or common area obtained by a RETRIEVE command from a started transaction.

Structure

This is the structure of the LISTEN service in COBOL language:

```
   01  CPT-ACM.
                COPY T09KCACM.

*****************************************************************
*                            CPT-ACM                            *
*     CICS PROGRAMMER'S TOOLKIT CONNECTION MANAGEMENT BLOCK     *
*---------------------------------------------------------------*
          05  ACMVERS          PIC   9(4) COMP VALUE 2.
              88  ACMVERSN     VALUE 2.
          05  ACMFUNC          PIC   9(4) COMP VALUE 0.
          05  ACMTOKEN         PIC   9(8) COMP VALUE 0.
          05  FILLER           PIC   9(8) COMP VALUE 0.
          05  FILLER           PIC   9(8) COMP VALUE 0.
          05  ACMRTNCD         PIC   9(8) COMP VALUE 0.
              88 ACM-RCOKAY    VALUE  0.
              88 ACM-WARNING   VALUE  1 THRU  15.
              88 ACM-ERROR     VALUE 16 THRU 255.
          05  ACMDGNCD         PIC   9(8) COMP VALUE 0.
          05  ACMSTATS         PIC   9(8) COMP VALUE 0.
          05  ACMTRACE         PIC   9(8) COMP VALUE 0.
          05  ACMQSEND         PIC   9(8) COMP VALUE 0.
          05  ACMMSEND         PIC   9(8) COMP VALUE 0.
          05  ACMQRECV         PIC   9(8) COMP VALUE 0.
          05  ACMMRECV         PIC   9(8) COMP VALUE 0.
          05  ACMTLSTN         PIC   9(8) COMP VALUE 0.
          05  ACMUCNTX         PIC   9(8) COMP VALUE 0.
          05  ACMTRNID         PIC   X(4) VALUE LOW-VALUES.
```

200801-024040-200100

```
05  FILLER          PIC   X(1) VALUE LOW-VALUES.
05  FILLER          PIC   X(1) VALUE LOW-VALUES.
05  FILLER          PIC   9(4) COMP VALUE 0.
05  ACMLPORT        PIC   9(4) COMP VALUE 0.
05  ACMRPORT        PIC   9(4) COMP VALUE 0.
05  ACMSRVCE        PIC  X(36) VALUE SPACES.
05  FILLER          PIC   X(1) VALUE LOW-VALUES.
05  FILLER          PIC   X(1) VALUE LOW-VALUES.
05  ACMOPTNS        PIC   9(4) COMP VALUE 0.
05  ACMLADDR        PIC   9(9) COMP VALUE 0.
05  ACMRADDR        PIC   9(9) COMP VALUE 0.
05  ACMLNAME        PIC X(255) VALUE SPACES.
05  FILLER          PIC   X(1) VALUE LOW-VALUES.
05  ACMRNAME        PIC X(255) VALUE SPACES.
05  FILLER          PIC   X(1) VALUE LOW-VALUES.
05  FILLER          PIC   9(4) COMP VALUE 0.
05  FILLER          PIC   9(4) COMP VALUE 0.
05  ACMTIMEO        PIC   9(8) COMP VALUE 0.
05  FILLER          PIC   9(8) COMP VALUE 0.
```

The following constants are contained in the `T09MAC` library member, `T09KCCON`:

```
   01  ACM-CONSTANTS.

*-----------------------------------------------------------------*
*   ACMSTATS          STATISTICS LOG REQUESTS                     *
*-----------------------------------------------------------------*
     * CONNECTION STATISTICS:
        05  ACMSTATS-CONN    PIC  9(4) COMP VALUE 1.
     * TERMINATION STATISTICS:
        05  ACMSTATS-TERM    PIC  9(4) COMP VALUE 2.

*-----------------------------------------------------------------*
*   ACMTRACE            TRACE LOG REQUESTS                        *
*-----------------------------------------------------------------*
     * TRACE ENTRY POINTS:
        05  ACMTRACE-NTRY    PIC  9(4) COMP VALUE 1.
     * TRACE ARGUMENTS:
        05  ACMTRACE-ARGS    PIC  9(4) COMP VALUE 2.
     * TRACE TRECV:
        05  ACMTRACE-RECV    PIC  9(4) COMP VALUE 4.
     * TRACE TSEND:
        05  ACMTRACE-SEND    PIC  9(4) COMP VALUE 8.
     * TRACE TERMINATION:
        05  ACMTRACE-TERM    PIC  9(4) COMP VALUE 16.
     * TRACE TAKE:
        05  ACMTRACE-PASS    PIC  9(4) COMP VALUE 32.
     * TRACE CLOSE:
        05  ACMTRACE-CLSE    PIC  9(4) COMP VALUE 64.
     * TRACE TPLERRORS:
        05  ACMTRACE-TERR    PIC  9(4) COMP VALUE 128.
     * TRACE TOKENS:
        05  ACMTRACE-TOKN    PIC  9(4) COMP VALUE 256.
     * TRACE TPL:
        05  ACMTRACE-TPL     PIC  9(4) COMP VALUE 512.
     * TRACE RELEASE:
        05  ACMTRACE-RLSE    PIC  9(4) COMP VALUE 1024.
    * TRACE STORAGE MANAGEMENT:
```

2C 024040-200100

```
           05  ACMTRACE-STOR     PIC   9(4) COMP VALUE 2048.
         * TRACE CLIENT DATA:
           05  ACMTRACE-CLTD     PIC   9(4) COMP VALUE 4096.

*-------------------------------------------------------------*
*   ACMOPTNS                PROCESSING OPTIONS                *
*-------------------------------------------------------------*
     * ISSUE SYNCPOINT FROM LISTEN
        05  ACMOPTNS-SYNC     PIC   9(4) COMP VALUE    1.
     * CLIENT-DATA LISTENER OPTION
        05  ACMOPTNS-LTRAN    PIC   9(4) COMP VALUE    2.
     * NO DNR NAME RESOLUTION
        05  ACMOPTNS-NODNR    PIC   9(4) COMP VALUE    4.
```

PARAMETER	DESCRIPTION
ACMVERS	Version Indicates the version number of the CPT argument list used by the calling program. This required field is set to 2 for this release of CPT. Default: 2
ACMFUNC	Function code Indicates the function or callable service ID requested by the application program. This field is not set by the application, but rather is initialized by the TRUE interface stub program. Default: None (generated by service stub)
ACMTOKEN	TCP connection token Specifies the token is created and returned by the CONNECT service. This token is used for all subsequent service calls for the client connection. Default: 0 (token returned)
ACMRTNCD	Return code Indicates the return code set by the CONNECT service. This value indicates the success or failure of the service. Default: 0
ACMDGNCD	Diagnostic code Indicates the diagnostic code received by the CONNECT service for a transport provider request. A detailed explanation of this value can be found in the transport provider's ***API Programmer's Reference Guide***. Default: 0
ACMSTATS	Specifies statistics logging options for the application program. The facility can be used for debugging and tuning during development. ◆ ACMSTATS-CONN Specifies that a message(s) be generated on establishing either a listen service or a data transfer connection. These messages are generated by the CPT LISTEN and CONNECT services. The message numbers associated with this option are CPT802I and CPT803I. ◆ ACMSTATS-TERM Specifies that a message(s) is to be generated on termination of an established connection. These messages are generated by the CPT CLOSE service. The message numbers associated with this option are CPT807I, CPT808I, and CPT809I. Default: 0 (no statistics logging)

200801-024040-200100

PARAMETER	DESCRIPTION
ACMTRACE	Specifies trace logging options for the application program. The facility can be used for debugging during development. ◆ ACMTRACE-NTRY – Specifies that a message be generated on entry to a CPT service routine. The message is generated by all CPT service routines. The message number associated with this option is CPT901I. ◆ ACMTRACE-ARGS – Specifies that a hex dump of the caller's arguments be generated on entry and exit of a CPT service routine. Messages are generated by the corresponding service routine. Message numbers associated with this option are CPT902I, CPT903I, CPT904I, CPT911I, CPT912, CPT921I, CPT922I, CPT924I, CPT925I, CPT930I, and CPT932I. ◆ ACMTRACE-RECV – Specifies that a hex dump of the transport provider's input (RECEIVE) data be logged. Messages are generated by the CPT RECV service. The message number associated with this option is CPT913I. ◆ ACMTRACE-SEND – Specifies that a hex dump of the transport provider's output (SEND) data be logged. Messages are generated by the CPT SEND service. The message number associated with this option is CPT914I. ◆ ACMTRACE-TERM – Specifies that a message be generated on termination of an CPT service routine. Messages are generated by all CPT service routines. The message number associated with this option is CPT908I. ◆ ACMTRACE-PASS – Specifies that a hex dump of resources related to a passed connection be logged. Messages are generated by the CPT GIVE and TAKE services. The message numbers associated with this option are CPT919I and CPT920I. ◆ ACMTRACE-CLSE – Specifies that a hex dump of resources related to a CLOSE processing of a connection be logged. Messages are generated by the CPT CLOSE service. The message number associated with this option is CPT906I ◆ ACMTRACE-TERR – Specifies that a hex dump of a transport provider API parameter list that fails successful completion be logged. The message number associated with this option is CPT400I. ◆ ACMTRACE-TOKN – Specifies that a hex dump of the connection token be logged. Messages are generated by all services routines on entry. The message number associated with this option is CPT909I. ◆ ACMTRACE-TPL – Specifies that a hex dump of the transport provider API parameter list be logged. The message numbers associated with this option are CPT917I and CTP935I. ◆ ACMTRACE-RLSE – Specifies that a message indicating a transport provider release indication be logged. Messages are generated by various CPT service routines. The message number associated with this option is CPT414I. ◆ ACMTRACE-STOR – Specifies that a hex dump of storage management argument be logged. Messages are generated by various CPT service routines. The message numbers associated with this option are CPT928I and CPT929I. ◆ ACMTRACE-CLTD – Trace transient data writes from the LISTEN service (used with the ACMOPTNS-LTRAN client-data option). The message number associated with this option is CPT918I. Default: 0 (no trace logging)
ACMQSEND	API send queue size Specifies the maximum number of uncompleted SEND requests that can be queued by the application to the transport provider (API). This value lets applications control output processing and can affect throughput rates. The value is negotiated with and can be modified by the transport provider. The total allocation for output processing is the product of the SEND queue and buffer size values and cannot exceed 61K. Default: 4

20-024040-200100

PARAMETER	DESCRIPTION
ACMMSEND	API send buffer size Specifies the maximum number of user data bytes that can be transferred by the application in a single SEND request to the transport provider (API). This value lets applications control output processing and can affect throughput rates. The value is negotiated with and can be modified by the transport provider. The total allocation for output processing is the product of the SEND queue and buffer size values and cannot exceed 61K. Default: 4096
ACMQRECV	API receive queue size Specifies the maximum number of uncompleted RECEIVE requests that can be queued by the application to the transport provider (API). This value lets applications control input processing and can affect throughput rates. The value is negotiated with and can be modified by the transport provider. The total allocation for input processing is the product of the RECEIVE queue and buffer size values and cannot exceed 61K. Default: 4
ACMMRECV	API receive buffer size Specifies the maximum number of user data bytes that can be transferred by the application, in a single RECEIVE request, to the transport provider (API). This value lets applications control input processing and can affect throughput rates. The value is negotiated with and can be modified by the transport provider. The total allocation for input processing is the product of the RECEIVE queue and buffer size values and cannot exceed 61K. Default: 1024
ACMTLSTN	Listen service token statistics Specifies the token used by the LISTEN service. This token is not available for data transfer. The only valid function that can be performed is a CLOSE request for long running active listeners. Generally, this value is not used by the application unless an explicit call to the CLOSE service is required. Read the description for ACMTOKEN (earlier in this table) for all other services. Default: 0 (token returned)
ACMUCNTX	One word of user context Specifies one arbitrary word of user context to be associated with the connection. The information provided is not interpreted by CPT, and is merely saved with other connection information. Default: 0 (no user context)
ACMTRNID	Listen start transaction ID A four-byte character string that the LISTEN service starts on successful establishment of a new connection. If TRANSID is specified, the LISTEN server loops for new connections and does not return to the calling program until CICS, CPT, or transport provider (API) termination. This field is optional and is not modified by the listen service. This field should not be specified if the ACMOPTNS_LTRAN option and ACMTIMEO value are specified. Default: None
ACMLPORT	Listen well-known service port Indicates the local transport layer address or port. This value represents the well-known port on which a server application will listen for connection requests. Either this value or the transport layer service name (ACMSRVCE) must be specified. This field is an unsigned positive integer with a maximum value of 65,534. The value must be unique for each server application. Default: None
ACMRPORT	Remote well-known service port Indicates the remote transport layer address or port. This value represents the client port number. This value is returned to the caller. This field is an unsigned positive integer with a maximum value of 65,534. Default: None

200801-024040-200100

PARAMETER	DESCRIPTION
ACMSRVCE	Transport layer service name Indicates the local transport layer service name. This string is mapped into a transport layer address or port, through internal DNR. The resolved value is the well-known port to which the client application connects. This field has a maximum length of 36 bytes. The value must be unique for each server application. This field is optional and is not modified by the LISTEN service. Default: None
ACMOPTNS	TCP connection initialization options ◆ ACMOPTNS-NODNR – DNR Suppression option. Skips internal DNR calls to resolve and return the remote IP address into an IP name in the ACMRNAME field. If an application is designed such that TCP connection establishment and release happens frequently, this option can save processing time. ◆ ACMOPTNS-LTRAN – Client-Data Listener option. Specifies that the Listen call will receive the input datastream to determine the transaction ID to be started. See ***Client-Data Listener Option*** for the required input formats. This option must be used with ACMTIMEO, and should not be used with ACMTRANID. ◆ ACMOPTNS-SYNC – Listen Syncpoint option. Issues a CICS syncpoint before starting any transaction from the LISTEN service. Default: None
ACMLADDR	Local IP host address Indicates the local host internet address. This field is an unsigned four-byte integer value. The local host internet address is updated on establishment of a server connection, and will be returned to the caller. Default: None
ACMRADDR	Remote IP host address Indicates the remote host internet address. Either this field or the remote host name (ACMRNAME) field must be specified. This field is an unsigned four-byte integer value. The remote host internet address is updated when a server connection is established, and is returned to the caller. Default: None
ACMLNAME	Local IP host name Indicates the local host internet name. This field is a 255-byte character string that is padded with blanks. The local host internet name is updated when a client connection is established, and is returned to the caller. Default: None
ACMRNAME	Remote IP host name Indicates the remote host internet name. Either this value or the remote IP address (ACMRADDR) field must be specified. This field is a 255-byte character string that is padded with blanks. The remote host internet name is updated when a server connection is established, and is returned to the caller. Default: None
ACMTIMEO	Client-Data Listener timeout values Specifies the maximum number of seconds that a Listener can wait to receive the client datastream when the ACMOPTNS_LTRAN option is specified. Default: 1 second

Network Considerations

The ACM is a common data structure used for both client and server connection initialization. There are common and unique values specified for a particular service request.

This table describes network considerations for COBOL API:

NAME	SERVER CONDITIONS FOR LISTEN	CLIENT CONDITIONS FOR CONNECT
ACMTLSTN	Listen token returned to user application.	
ACMTRNID	Listen START transaction ID.	
ACMLPORT	Local server or listening transport provider well-known port selected by user application.	Local assigned transport provider port returned to user application.
ACMRPORT	Remote client transport provider port returned to user application.	Remote server transport provider well-known port selected by user application.
ACMSRVCE	Local server transport provider service name selected by user application.	Remote server transport provider service name selected by user application.
ACMRADDR	Remote IP host address returned to user application.	Remote IP host address selected or returned to user application.
ACMLNAME	Local IP host name returned to user application.	Remote IP host name returned to user application.
ACMRNAME	Remote IP host name returned to user application.	Remote IP host name selected or returned to user application.
ACMTIMEO	Client-Data Listener timeout value.	

200801-024040-200100

Completion Information

Completion of a request to the LISTEN service depends on the arguments selected. If no transaction ID is specified, the LISTEN service returns control to the calling program when connection with a client is established. The caller's argument list is updated with various information related to the new connection. If a transaction ID is specified, the LISTEN service does not return control to the calling program until a failure is detected. The caller's argument list is generally not updated, with exception to the return code information.

The LISTEN service initializes the server environment with the transport provider (API), waits for a connection request, establishes a connection with the client, and updates connection information within the ACM. Establishing a listening connection and a client connection are represented by storage and are referred to hereafter as the token. Establishing a client connection updates the ACM with information relative to the connection. The information is returned to the user or is passed to the data processing transaction.

The server application contains two tokens representing endpoints to the transport provider. The first token (ACMTOKEN) represents the client connection and is used for data transfer. The other token (ACMTLSTN) represents the listening connection. This listening connection can only be referenced within the CPT CLOSE service. This lets an explicit ability close a server or listening connection. All other CPT services performed with the LISTEN token fail with an invalid token. Implicit clean-up of the LISTEN token is provided by the TRUE interface. Therefore, an explicit call to the CLOSE service is not required.

When the LISTEN service is initiated without a transaction ID, control is returned to the calling program when a connection with a client is established. The caller's argument list is updated with information related to the new connection. The local and remote port, IP address, and host names are resolved. Negotiated transport provider SEND and RECEIVE buffering values are returned. The ACM return code (ACMRTNCD) must be checked to determine the success or failure of LISTEN service. A zero (0) return code indicates a successful client connection is established.

When the LISTEN service is initiated with a transaction ID, it operates as a CICS long running task. The LISTEN service establishes client connections and starts a data processing transaction. The data processing transaction receives a copy of the connection management argument. The data transfer or client connection token is derived from the ACMTOKEN field. After the new transaction is initiated the LISTEN service continues waiting for new client connections. The LISTEN service continues to listen and start client connections until an error occurs.

The return and diagnostic codes should be interrogated to determine the reason for failure. Errors indicating CPT, the transport provider (API), or CICS termination are minor. Errors should be interrogated for level of severity.

Return Codes

The LISTEN service return and diagnostic codes indicate the result of the execution. These values are in the ACMRTNCD and ACMDGNCD within the ACM. The diagnostic code is optional and indicates the transport provider return code. Read **Appendix C – MESSAGES AND CODES** in the ***CICS Programmer's Toolkit Installation and Administration Guide*** for the return code cross reference table.

This table describes the LISTEN service return codes:

RETURN	DIAGNOSTIC CODE	DESCRIPTION
CPTIRCOK		Successful.
CPTEVERS		Control block version number is not supported.
CPTECONN	Yes	Requested host/service/port is not found.
CPTEPROT		Specified protocol is not supported.
CPTENAPI	Yes	Transport provider API is not available.
CPTETERM	Yes	Environment is being terminated.
CPTERLSE	Yes	Release indication.
CPTEDISC	Yes	Disconnect indication.
CPTEPRGE	Yes	CPT Interface terminating.
CPTEINTG	Yes	Transport provider API integrity error.
CPTEENVR	Yes	Transport provider API environment error.
CPTEFRMT	Yes	Transport provider API format error.
CPTEPROC	Yes	Transport provider API procedure error.
CPTABEND		Abnormal exception occurred.
CPTEOTHR	Yes	An undefined exception occurred.

200801-024040-200100

Usage Information

The LISTEN service lets user-written application programs implement TCP/IP server facilities. Server applications passively wait, then establish connections with single- or multi-thread support. The LISTEN service generalized parameter list (ACM) describes the application's communications requirements as well as information related to established connections. The ACM contains fields initialized by both a user application and by the LISTEN service, on completion.

There are required and optional fields initialized by a user or calling application. The ACM version number and the transport provider local port or service name are required. The selection of port or service name defines the server well-known port address. Optional fields control data transfer buffering, statistics, tracing, and subtask initialization.

When the LISTEN service completes or the data processing task executes, the ACM contains information related to the established connection. A token that identifies the connection is returned in the ACM, and must be used in all subsequent requests that refer to the connection. The application program should make no assumptions regarding the format of a token, other than it is an unsigned, full word value.

Information related to the negotiated buffer values, host names, host addresses, and transport provider addresses are returned in the ACM.

The version number (ACMVERS) indicates the CPT release level in which this user application program is written. This required field must be set to ACMVERS2 and is validated by the LISTEN service before processing the request.

The function code (ACMFUNC) indicates the CPT callable service ID. The field is initialized by the CPT service stub program. The function code identifies argument lists within the error or trace logs, and dump analysis.

The transport provider port number (ACMPORT) or service name (ACMSRVCE) is required. These fields identify the well-known port to which a client initiates a connection request. The port number has precedence over service name. This implies that the service name field is only used if a port number is not specified.

The transaction ID field (ACMTRNID) identifies a data processing task. This is an optional field that causes the LISTEN service to execute continuously. The LISTEN service starts a new transaction after a client connection is established, then waits for additional connection requests. An updated ACM is passed to the data processing task. Control is not returned to the calling program until an error occurs. The return code indicates the reason for the failure. Errors indicating the transport provider, CICS, or CPT termination are acceptable. Errors indicating port in use, API unavailable, or program checks should be investigated.

User application programs can control CPT and transport provider data transfer buffering. The ACMQSEND, ACMMSEND, ACMQRECV, and ACMMRECV specify the number and size of buffers allocated. The SEND and RECEIVE buffers are allocated on initial entry into either the SEND or RECEIVE service. The corresponding values used by the SEND and RECEIVE services are independent of each other. The SEND service multiplies the queue and buffer values to determine output storage requirements. The RECEIVE service performs a

similar function to determine input storage requirements. The product of the queue and buffer values cannot exceed 61,440. CPT requires some additional storage use to manage these buffers. This extra storage is included in the allocation.

The CPT SEND service uses the ACMQSEND value, and the CPT RECEIVE service uses the ACMQRECV value. These values indicate the maximum number of uncompleted SEND or RECEIVE requests that can be queued by the application to the transport provider. The CPT data transfer services schedule asynchronous transport provider or API requests on the caller's behalf. These API requests must be completed before they can be used again. If the queue values are too small the CPT data transfer service may block the caller's request and schedule a wait within the service routines. If the queue values are too large the user application may be wasting storage.

The CPT SEND service uses the ACMMSEND value; the CPT RECEIVE service uses the ACMMRECV value. These values indicate the maximum number of user data bytes that can be transferred by the application in a single SEND or RECEIVE request to the transport provider. The user application is not limited to these values within the data transfer services. However, it is important to note that multiple transport provider or API requests are issued to complete the caller's request. Information on queue and buffer size are in the sections **RECEIVE** on page 4-52 and **SEND** on page 4-61.

Initially, the tuning of data transfer storage may not be a concern. However, the ability to control storage allocation can prove beneficial to the application or CICS region. Queue size can increase data transfer throughput. You should consider enabling the statistics option to gather CPT statistical information. This information can set the SEND or RECEIVE queue and buffer size values.

The LISTEN service can modify the data transfer buffer allocation values. These values are negotiated with the transport provider and, depending on the site configuration, can be reduced. Any application dependent on these values should check them on return. These values are generally not modified when giving reasonable numbers. However, it is advisable to check with the site administrator for the maximum values of the API transport services.

A number of arguments are not set by the calling application, but are returned to the caller. These values represent information related to the client connection and can be used by the application. The local port, host name, and IP address are returned as well as the client's corresponding values. An ACM is passed the started transaction when a TRANSID is specified in the caller's LISTEN argument list.

200801-024040-200100

Client-Data Listener Option

The option ACMOPTNS_LTRAN is used in conjunction with ACMTIMEO and is mutually exclusive of the use of the ACMTRNID field. ACMOPTNS_LTRAN indicates to the LISTEN service that the connecting client application will specify what server functions to execute. When the LISTEN service receives a CONNECT request and ACMOPTNS_LTRAN is specified, it uses a partial record timed RECEIVE (see RECEIVE service options) to get the client's data. It uses ACMTIMEO to know how long to wait for the client data which can be in any of these formats:

```
TRAN
TRAN,UUUUUUUUUUUU
TRAN,UUUUUUUUUUUU,IC,HHMMSS
TDQN,UUUUUUUUUUUU,TD
TRAN,,IC,HHMMSS
TDQN,,TD
```

PARAMETER	DESCRIPTION
TRAN	A 1- to 4-character transaction ID to start, passing CLNT-PARM to the started transaction
UUUUUUUUUUUU	A 1- to 35-byte user data and is passed to the started transaction or written to the transient data queue in the field CP-DATA.
IC	Specifies that TRAN is to be started in HHMMSS; if left blank, startup is immediate.
HHMMSS	Hours, minutes, and seconds for IC option.
TD	Indicates that CLNTPARM will be written into the transient data queue, TDQN.

```
*****************************************************************
*              COBOL CLNT-PARM DESCRIPTION                      *
*---------------------------------------------------------------*
   01  CLNT-PARM.
   05  CP-TOKEN          PIC    9(8) COMP.
   05  FILLER            PIC    X(16).
   05  CP-DATA           PIC    X(36).
   05  CP-ADDRESS.
      10 CP-DOMAIN       PIC    9(4) COMP.
      10 CP-RPORT        PIC    9(4) COMP.
      10 CP-RADDR        PIC    9(4) COMP.
   05  FILLER            PIC    X(8).
```


Note: Using this option puts a listener at risk of being tied up until the client actually sends the data. Set ACMTIMEO with this fact in mind. The new trace flag, ACMTRACE_CLTD, can optionally be specified to trace the CLNT-PARM written to TDQN transient data queue via CPT918I.

Example:

This example establishes a server connection, processes data, and closes the connection. The server listens on well-known port 1234. The token is loaded from the ACM and used by all of the following CPT service requests. The return code is checked to determine `LISTEN` service completion status.

This sample program is generally not the preferred server model. The problem is that after returning from the `LISTEN` service the application blocks additional incoming connection requests. A better example of this facility is show in the following example. This single-threaded server model is only suitable for connections of a very short time duration.

```
WORKING-STORAGE SECTION.
01  CPT-ACM    COPY T09KCACM.
01  CPT-ADT    COPY T09KCADT.
01  CPT-ACL    COPY T09KCACL.
01  CPT-RCS    COPY T09KCRCS.

PROCEDURE DIVISION.

    PERFORM UNTIL SERVER-OK

*
*       CPT LISTEN Connection Management initialization and request
*
        MOVE 1234 TO ACMLPORT.
        CALL 'T09FLSTN' USING CPT-ACL.
        IF NOT ACM-RCOKAY
*           Process and log LISTEN service error
            SET SERVER-OK TO TRUE
        ELSE
            MOVE ACMTOKEN TO ADTTOKEN ACLTOKEN.
*
*           Application and CPT data transfer (SEND/RECEIVE) processing
*
            PERFORM UNTIL NOT ADT-RCOKAY
    .
    .
    .
            END-PERFORM.
*
*       CPT Data Transfer Connection Release processing

            .
            .
            .
        END-IF.
    END-PERFORM.

*
* Terminate Transaction
*
    EXEC CICS RETURN
    END-EXEC.
```

Example:

This example is a multi-threaded server application. This task initiates client connections and starts a data processing transaction. The server listens on the port mapped into service name ECHO. The token is loaded from the ACM and is passed to the data transfer transaction. The return code is checked to determine LISTEN service completion status.

This sample program differs from the first example in that data transfer is not performed by the listening transaction, but by a different transaction. This makes a more efficient server program. The server application respond more quickly to new connection requests because it is not involved in the task of data transfer or connection management after the initialization connection.

This example does not use the optional GIVE facility management service. The GIVE service would be beneficial if this task was expected to terminate before the data processing transaction initiated. However, since this task is not expected to terminate, any abnormal termination would release all CPT connections currently associated with this task.

```
WORKING-STORAGE SECTION.
01  CPT-ACM    COPY T09KCACM.
01  CPT-ADT    COPY T09KCADT.
01  CPT-RCS    COPY T09KCRCS.

PROCEDURE DIVISION.

    PERFORM UNTIL SERVER-NOT-OK

*
*       CPT LISTEN Connection Management initialization and request
*
        MOVE 'ECHO' TO ACMSRVCE.
        CALL 'T09FLSTN' USING CPT-ACM.
        IF NOT ACM-RCOKAY
*           Process and log LISTEN service error
            SET SERVER-NOT-OK TO TRUE
        ELSE
*
*           CPT GIVE Facility Management Service
*
            EXEC CICS START TRANSID(trans id) FROM(CPT-ACM) LENGTH()
            END-EXEC.
        END-IF.
    END-PERFORM.

*
* Terminate Transaction
*
    EXEC CICS RETURN
    END-EXEC.
```

Example:

This multi-threaded server example listens for connections resolved to the transport service name DISCARD. A TRANSID is specified that causes the LISTEN service to start a data processing transaction. Control is not returned to this program until an error has occurred. The ACM-RTNCD is checked to determine LISTEN service request completion status.

This sample program differs from the second example in that a START command for a new transaction is performed by the LISTEN service and not by the user application. Also, control is not returned to the calling application until a failure occurs. Generally, this failure is due to termination of CICS, CPT, or the transport provider (API).

In this example, transaction SRV3 is started automatically by the LISTEN service for each connection established on the DISCARD port.

```
WORKING-STORAGE SECTION.
01  CPT-ACM    COPY T09KCACM.
01  CPT-ADT    COPY T09KCADT.
01  CPT-RCS    COPY T09KCRCS.

PROCEDURE DIVISION.

*
*   CPT LISTEN Connection Management initialization and request
*
    MOVE 'DISCARD' TO ACMSRVCE.
    MOVE 'SVR3' TO ACMTRNID.
    CALL 'T09FLSTN' USING CPT-ACL.
    IF NOT ACM-RCOKAY
*       Process and log LISTEN service error
    END-IF.
*
* Terminate Transaction
*
    EXEC CICS RETURN
    END-EXEC.
```

200801-024040-200100

RCVFROM

The RCVFROM service lets you develop connectionless client and server applications. This service is UDP only. The RCVFROM service provides these basic functions:

- Establishes a UDP server endpoint represented by a new token and starts receiving datagrams on a user-specified well-known port. Indicate this function to the RCVFROM service by passing an ADTTOKEN equal to zero. RCVFROM then creates all the internal control blocks and the RCVFROM buffer queue. Even though the SENDTO buffer queue is not allocated for this endpoint (token) until the SENDTO service is called, the SENDTO buffer size and number must be specified at this time because they are negotiated with the transport provider and recorded in the internal CPT control blocks at endpoint creation time. Upon return from the RCVFROM service, ADTTOKEN contains the token value to be passed to subsequent RCVFROM and SENDTO service calls.
- Receives a datagram at a previously established UDP endpoint represented by an existing token. This functionality makes the RCVFROM service call just a data transfer call that can be used by a client or server application. The RCVFROM buffer queue is only allocated upon the first call to the RCVFROM service, whether or not ADTTOKEN is equal to zero.

UDP tokens created with the RCVFROM or SENDTO services cannot be passed to the TCP-only services, CONNECT, LISTEN, SEND, and RECEIVE. All other CPT service calls are available to UDP applications.

The non-blocking option of the RCVFROM service (ADTOPTNS-NBLKR), allows applications to be developed that can poll a well-known UDP port or send to a remote UDP server and then make a predetermined number of RCVFROM calls to get back a response. Given the general unreliable nature of UDP, not blocking on a RCVFROM call can build in some flexibility with regards to handling lost datagrams.

This table describes RECEIVE service arguments:

COPY NAME	SIZE	CREATED BY
T09KCADT	644 (X'284')	User application

Structure

This is the structure of the RECEIVE service in COBOL language:

```
*  01  CPT-ADT.
                COPY T09KCADT.

*****************************************************************
*                          CPT-ADT                              *
*     CICS PROGRAMMER'S TOOLKIT DATA TRANSFER CONTROL BLOCK     *
*---------------------------------------------------------------*
          05  ADTVERS          PIC   9(4) COMP VALUE 2.
              88  ADTVERSN     VALUE 2.
```

```
 05  ADTFUNC          PIC   9(4) COMP VALUE 0.
 05  ADTTOKEN         PIC   9(8) COMP VALUE 0.
 05  ADTBUFFA         POINTER    VALUE NULL.
05  ADTINCRA REDEFINES ADTBUFFA PIC 9(8) COMP.
 05  ADTBUFFL         PIC   9(8) COMP VALUE 0.
 05  ADTRTNCD         PIC   9(8) COMP VALUE 0.
     88  ADT-RCOKAY                 VALUE  0.
     88  ADT-NO-MORE-DATA           VALUE 65.
     88  ADT-WARNING               VALUE  1 THRU 15.
     88  ADT-ERROR                VALUE 16 THRU 255.
 05  ADTDGNCD         PIC   9(8) COMP VALUE 0.
 05  ADTSTATS         PIC   9(8) COMP VALUE 0.
 05  ADTTRACE         PIC   9(8) COMP VALUE 0.
 05  ADTQSEND         PIC   9(8) COMP VALUE 0.
 05  ADTMSEND         PIC   9(8) COMP VALUE 0.
 05  ADTQRECV         PIC   9(8) COMP VALUE 0.
 05  ADTMRECV         PIC   9(8) COMP VALUE 0.
 05  ADTTIMEO         PIC   9(8) COMP VALUE 0.
 05  FILLER           PIC   9(8) COMP VALUE 0.
 05  FILLER           PIC   9(8) COMP VALUE 0.
 05  FILLER           PIC   9(8) COMP VALUE 0.
 05  FILLER           PIC   9(8) COMP VALUE 0.
 05  ADTLPORT         PIC   9(4) COMP VALUE 0.
 05  ADTRPORT         PIC   9(4) COMP VALUE 0.
 05  FILLER           PIC   9(4) COMP VALUE 0.
 05  ADTSRVCE         PIC  X(36) VALUE ' '.
 05  ADTSEPN          PIC   9(4) COMP VALUE 0.
     88 ADTSEPN-1                 VALUE 1.
     88 ADTSEPN-2                 VALUE 2.
 05  ADTSEP1          PIC   X(1) VALUE ' '.
 05  ADTSEP2          PIC   X(1) VALUE ' '.
 05  FILLER           PIC   9(4) COMP VALUE 0.
 05  ADTLADDR         PIC   9(8) COMP VALUE 0.
 05  ADTRADDR         PIC   9(8) COMP VALUE 0.
 05  ADTLNAME         PIC X(255) VALUE ' '.
 05  FILLER           PIC   X(1) VALUE LOW-VALUE.
 05  ADTRNAME         PIC X(255) VALUE ' '.
 05  FILLER           PIC   X(1) VALUE LOW-VALUE.
 05  ADTUCNTX         PIC   9(8) COMP VALUE 0.
 05  ADTOPTNS         PIC   9(8) COMP VALUE 0.
```

The following constants are contained in the T09MAC library member, T09KCCON:

```
01  ADT-CONSTANTS.

*-------------------------------------------------------------*
*  ADTSTATS AND ADTTRACE VALUES ARE THE SAME AS DEFINED FOR   *
*  THE ACM, AND ARE SET ONLY FOR UDP CALLS, RCVFROM AND SENDTO*
*  TO SET THESE FIELDS, USE THE EQUIVALENT ACM-CONSTANT.      *
*-------------------------------------------------------------*
*   ADTOPTNS        SPECIAL SEND AND RECEIVE PROCESSING MODES.*
*-------------------------------------------------------------*
     * SEPARATOR CHARACTER SEND/RECEIVE
         05  ADTOPTNS-TYPSP    PIC  9(8) COMP VALUE    1.
      * LENGTH (LL) SEND/RECEIVE
         05  ADTOPTNS-TYPLL    PIC  9(8) COMP VALUE    2.
      * BLOCK ON SEND
```

```
    05 ADTOPTNS-BLCKS    PIC  9(8) COMP VALUE    4.
* TIMED PARTIAL RECEIVE
    05 ADTOPTNS-TMPRT    PIC  9(8) COMP VALUE    8.
* TIMED FULL BLOCK RECEIVE
    05 ADTOPTNS-TMRCV    PIC  9(8) COMP VALUE   16.
* DO NOT BLOCK ON RECEIVE/RCVFROM
    05 ADTOPTNS-NBLKR    PIC  9(8) COMP VALUE   64.
* DO DNR NAME RESOULTION IN UDP
    05 ADTOPTNS-DODNR    PIC  9(8) COMP VALUE  128.
* LEAVE SEPARATOR OR LENGTH BYTES WITH DATA
    05 ADTOPTNS-NOSTP    PIC  9(8) COMP VALUE 16384.
* VECTOR LIST FLAG
    05 ADTOPTNS-FVLST    PIC  9(8) COMP VALUE 32768.
```

PARAMETER	DESCRIPTION
ADTVERS	Version Indicates the CPT version number of the argument used by the calling program. This required field is set to 2 for this release of CPT. Default: 2
ADTFUNC	Function code Indicates the function or callable service ID requested by the application program. This field is not set by the application, but is initialized by the TRUE interface stub program. Default: None
ADTTOKEN	Data transfer token Specifies a token that represents a UDP endpoint. If the ADT is being passed in a call to either the RCVFROM or SENDTO service, then the token can be zero, indicating to either service, to first create a token before sending or receiving a datagram. If the token is not zero, then it must be a token created previously by either the RCVFROM or SENDTO service. It is not necessary or efficient to create a token every time a CICS transaction calls the UDP data transfer services. It is an error to pass a TCP token to the UDP data transfer service routines, RCVFROM and SENDTO. Conversely it is an error to pass a UDP token to the TCP data transfer routines, RECEIVE and SEND. Default: 0
ADTBUFFA	User data address Indicates the storage address into which the UDP datagram is received (RCVFROM service). This is a contiguous segment of storage accessible to the user task. The content of all user data is application dependent, and not interpreted by either CPT nor the transport provider. The storage area can be aligned on any boundary convenient for the application program. Default: 0
ADTINCRA	Redefines ADTBUFFA Allows for computational updating of the user data address.
ADTBUFFL	User data length Indicates the length (in bytes) of the buffer specified in ADTBUFFA which is to receive (RCVFROM service) the UDP datagram. If the incoming datagram does not fit into ADTBUFFA for a length of ADTBUFFL, then the warning, CPTWNEOM will be passed back to the caller in ADTRTNCD, indicating that more RCVFROM calls will be required to get the entire datagram. ADTBUFFL will indicate the actual length returned in ADTBUFFA. It is an error to call the RCVFROM service with an ADTBUFFL of zero.
ADTRTNCD	Return code Indicates the return code set by the RCVFROM service. This value is returned and indicates the success or failure of the service. Default: 0

PARAMETER	DESCRIPTION
ADTDGNCD	Diagnostic code Indicates the diagnostic code set by the service request. This value generally indicates a transport provider return code. Default: 0
ADTSTATS	Specifies statistics logging options for the application program. The facility can be used for debugging and tuning during development. ◆ ADTSTATS-CONN Specifies that a message(s) be generated on establishing either a listen service or a data transfer connection. These messages are generated by the CPT LISTEN and CONNECT services. The message numbers associated with this option are CPT802I and CPT803I. ◆ ADTSTATS-TERM Specifies that a message(s) is to be generated on termination of an established connection. These messages are generated by the CPT CLOSE service. The message numbers associated with this option are CPT807I, CPT808I, and CPT809I. Default: 0 (no statistics logging)

200801-024040-200100

PARAMETER	DESCRIPTION
ADTTRACE	Specifies trace logging options for the application program. The facility can be used for debugging during development. ◆ ADTTRACE-NTRY – Specifies that a message be generated on entry to a CPT service routine. The message is generated by all CPT service routines. The message number associated with this option is CPT901I. ◆ ADTTRACE-ARGS – Specifies that a hex dump of the caller's arguments be generated on entry and exit of a CPT service routine. Messages are generated by the corresponding service routine. Message numbers associated with this option are CPT902I, CPT903I, CPT904I, CPT911I, CPT912, CPT921I, CPT922I, CPT924I, CPT925I, CPT930I, and CPT932I. ◆ ADTTRACE-RECV – Specifies that a hex dump of the transport provider's input (RECEIVE) data be logged. Messages are generated by the CPT RECV service. The message number associated with this option is CPT913I. ◆ ADTTRACE-SEND – Specifies that a hex dump of the transport provider's output (SEND) data be logged. Messages are generated by the CPT SEND service. The message number associated with this option is CPT914I. ◆ ADTTRACE-TERM – Specifies that a message be generated on termination of an CPT service routine. Messages are generated by all CPT service routines. The message number associated with this option is CPT908I. ◆ ADTTRACE-PASS – Specifies that a hex dump of resources related to a passed connection be logged. Messages are generated by the CPT GIVE and TAKE services. The message numbers associated with this option are CPT919I and CPT920I. ◆ ADTTRACE-CLSE – Specifies that a hex dump of resources related to a CLOSE processing of a connection be logged. Messages are generated by the CPT CLOSE service. The message number associated with this option is CPT906I ◆ ADTTRACE-TERR – Specifies that a hex dump of a transport provider API parameter list that fails successful completion be logged. The message number associated with this option is CPT400I. ◆ ADTTRACE-TOKN – Specifies that a hex dump of the connection token be logged. Messages are generated by all services routines on entry. The message number associated with this option is CPT909I. ◆ ADTTRACE-TPL – Specifies that a hex dump of the transport provider API parameter list be logged. The message numbers associated with this option are CPT917I and CTP935I. ◆ ADTTRACE-RLSE – Specifies that a message indicating a transport provider release indication be logged. Messages are generated by various CPT service routines. The message number associated with this option is CPT414I. ◆ ADTTRACE-STOR – Specifies that a hex dump of storage management argument be logged. Messages are generated by various CPT service routines. The message numbers associated with this option are CPT928I and CPT929I. Default: 0 (no trace logging)
ADTQSEND	API send queue size (used when ADTTOKEN=0) Specifies the maximum number of uncompleted SENDTO requests that can be queued by the application to the transport provider (API). This value lets applications control output processing and can affect throughput rates. The value is negotiated with the transport provider and can be modified by the transport provider. The total allocation for output processing is the product of the SENDTO queue and buffer size values and cannot exceed 61K. Default: 4

PARAMETER	DESCRIPTION
ADTMSEND	API send buffer size (used when ADTTOKEN=0) Specifies the maximum number of user data bytes that can be transferred by the application in a single SENDTO request to the transport provider (API). This value lets applications control output processing and can affect throughput rates. The value is negotiated with the transport provider and can be modified by the transport provider. The total allocation for output processing is the product of the SENDTO queue and buffer size values and cannot exceed 61K. Default: 4096
ADTQRECV	API receive queue size (used when ADTTOKEN=0) Specifies maximum number of uncompleted RCVFROM requests that can be queued by the application to the transport provider (API). This value lets applications control input processing and can affect throughput rates. The value is negotiated with the transport provider and can be modified by the transport provider. The total allocation for input processing is the product of the RCVFROM queue and buffer size values and cannot exceed 61K. Default: 4
ADTMRECV	API receive buffer size (used when ADTTOKEN=0) Specifies the maximum number of user data bytes that can be transferred by the application in a single RCVFROM request to the transport provider (API). This value lets applications control input processing and can affect throughput rates. The value is negotiated with the transport provider and can be modified by the transport provider. The total allocation for input processing is the product of the RCVFROM queue and buffer size values and cannot exceed 61K. Default: 4096
ADTTIMEO	RECEIVE time out value Not used by the RCVFROM service Default: 0
ADTLPORT	Local well-known service port (used when ADTTOKEN=0) Indicates the local transport layer port that the calling application will be receiving (RCVFROM) datagrams on. If the SENDTO service creates the token, this port number is assigned by the transport layer and returned to the caller. if the RCVFROM service creates the token, this is the well-known port requested by the caller. If the RCVFROM service is creating the token, this value or the transport layer service name (ADTSRVCE) must be specified. This field is an unsigned positive integer with a maximum value of 65,534. The value must be unique for each server application. Default: None
ADTRPORT	Remote port Indicates the remote transport layer port on which the incoming datagram originated. This value is returned to the caller of the RCVFROM service and may be different for each datagram received. This field is an unsigned positive integer with a maximum value of 65,534. Default: None
ADTSRVCE	Transport layer service name (used when ADTTOKEN=0) Indicates the local transport layer service name. This string is mapped into a transport layer address or port, through internal DNR. The resolved value is the well-known port to which the client application sends (SENDTO) UDP datagrams. This field has a maximum length of 36 bytes. The value must be unique for each server application. This field is optional and is not modified by the RCVFROM service. Default: None
ADTSEPN	Number of separator characters for option ADTOPTNS-TYPSP. Not used in the RCVFROM service. Default: None
ADTSEPN-1	First or only spaceport character for option ADTOPTNS-TYPSP. Not used in the RCVFROM service. Default: None

200801-024040-200100

PARAMETER	DESCRIPTION
ADTSEPN-2	Second character or separator sequence for option ADTOPTNS-TYPSP Not used in the RCVFROM service. Default: None
ADTLADDR	Local IP host address Indicates the local host internet address. This field is an unsigned four-byte integer value. The local host internet address is updated on establishment of a server connection, and will be returned to the caller. Default: None
ADTRADDR	Remote IP host address Indicates the remote host internet address. Either this field or the remote host name (ADTRNAME) field must be specified. This field is an unsigned four-byte integer value. The remote host internet address is updated when a server connection is established, and is returned to the caller. Default: None
ADTLNAME	Local IP host name Indicates the local host internet name. This field is a 255-byte character string that is padded with blanks. The local host internet name is updated when a client connection is established, and is returned to the caller. Default: None
ADTRNAME	Remote IP host name Indicates the remote host internet name. Either this value or the remote IP address (ADTRADDR) field must be specified. This field is a 255-byte character string that is padded with blanks. The remote host internet name is updated when a server connection is established, and is returned to the caller. Default: None
ACMUCNTX	One word of user context Specifies one arbitrary word of user context to be associated with the connection. The information provided is not interpreted by CPT, and is merely saved with other connection information. Default: 0 (no user context)
ADTOPTNS	Specifies data transfer options These are the ADT options that apply to UDP data transfer requests: ◆ ADTFDNR – Execute internal DNR calls during UDP data transfer service routine calls (RCVFROM and SENDTO) to resolve remote IP addresses into IP names in the ADTRNAME field. ◆ ADTNBLKR – Do not block on a call to the RCVFROM service. If no datagrams are currently available, the return code, CPTWBLCK, will be returned in ADTRTNCD. Default: None These options can be toggled on every UDP data transfer call even if the caller is using the same token.

20 9024040-200100

Network Considerations

The ADT is a common data structure used for both client and server UDP applications. There are common and unique values specified for a particular service request.

The table describes network considerations for COBOL API:

NAME	SERVER CONDITIONS FOR RECVFROM	CLIENT CONDITIONS FOR SENDTO
ADTLPORT	Local server well-known port selected by user application.	Local assigned transport provider port returned to user application.
ADTRPORT	Remote client transport provider port returned to user application.	Remote server transport provider well-known port selected by user application.
ADTSRVCE	Local server transport provider service name selected by user application.	Remote server transport provider service name selected by user application.
ADTRADDR	Remote IP host address returned to user application.	Remote IP host address selected by or returned to user application. The client must specify this field or ADTRNAME.
ADTLNAME	Local IP host name returned to user application.	Local IP host name Returned to user application.
ADTRNAME	Remote IP host name returned to user application only if ADTOPTNS-DODNR is specified.	Remote IP host name selected by or returned to the user application. The client must specify this field or ADTRADDR. If ADTRADDR is used ADTRNAME will only be returned if ADTOPTNS-DODNR is specified.

200801-024040-200100

Return Codes

The RCVFROM service return and diagnostic codes indicate the result of the execution. These values are in the ADTRTNCD and ADTDGNCD within the ADT. The diagnostic code is optional and generally indicates a transport provider return code. Read **Appendix C – MESSAGES AND CODES** in the ***CICS Programmer's Toolkit Installation and Administration Guide*** for the return code cross reference table.

This table describes the RCVFROM service return codes:

RETURN	DIAGNOSTIC CODE	DESCRIPTION
CPTIRCOK		Successful.
CPTEVERS		Control block version number is not supported.
CPTETOKN		Specified token is not valid.
CPTEBUFF		Buffer address and/or length is invalid.
CPTENAPI	Yes	Transport provider API is not available.
CPTETERM	Yes	Environment is being terminated.
CPTERLSE	Yes	Release indication.
CPTEDISC	Yes	Disconnect indication.
CPTEINTG	Yes	Transport provider API integrity error.
CPTEENVR	Yes	Transport provider API environment error.
CPTEFRMT	Yes	Transport provider API Format error.
CPTEPROC	Yes	Transport provider API procedure error.
CPTABEND		Abnormal exception occurred.
CPTEOTHR	Yes	An undefined exception occurred.

Example:

This example receives data from a remote host. The token is loaded from the ACM and used by all of the following CPT service requests. The return code is checked to determine RCVFROM service completion status.

```
WORKING-STORAGE SECTION.
01 CPT-ADT    COPY T09KCADT.
              COPY T09KCRCS.

LINKAGE SECTION.
01  MESSAGE PIC X(1023).

PROCEDURE DIVISION.

*
* Identify Service
*
    .
```

20 024040-200100

```
      .
      .
      MOVE 1980 TO ADTLPORT.

*
* Application and CPT RECEIVE Data Transfer processing
*
      PERFORM UNTIL NOT ADT-RCOKAY
          SET ADTBUFFA TO ADDRESS OF MESSAGE
          SET ADTBUFFL TO LENGTH OF MESSAGE
          CALL 'T09RCFR' USING CPT-ADT
          IF NOT ADT-RCOKAY
*             Process and log RECEIVE service error
          END-IF
      END-PERFORM.

*
* Terminate Transaction
*
      EXEC CICS RETURN
      END-EXEC.
```

200801-024040-200100

RECEIVE

The RECEIVE service receives data from a peer transport user connected to an endpoint. The RECEIVE service receives data as input on a connection-mode (TCP) endpoint only.

To invoke the RECEIVE service, a user application must first build an Argument for Data Transfer (ADT) and then issue a call to the RECEIVE routine. The ADT contains the version number, connection token, user buffer address, and length. When the RECEIVE service completes, the buffer length field is updated to reflect the amount of data processed by the RECEIVE service.

This table describes RECEIVE service arguments:

COPY NAME	SIZE	CREATED BY
T09KCADT	644 (X'284')	User application

Structure

This is the structure of the RECEIVE service in COBOL language:

```
*  01  CPT-ADT.
               COPY T09KCADT.

*****************************************************************
*                          CPT-ADT                              *
*     CICS PROGRAMMER'S TOOLKIT DATA TRANSFER CONTROL BLOCK     *
*---------------------------------------------------------------*
         05  ADTVERS          PIC  9(4) COMP VALUE 2.
             88  ADTVERSN     VALUE 2.
         05  ADTFUNC          PIC  9(4) COMP VALUE 0.
         05  ADTTOKEN         PIC  9(8) COMP VALUE 0.
         05  ADTBUFFA         POINTER   VALUE NULL.
        05  ADTINCRA REDEFINES ADTBUFFA PIC 9(8) COMP.
         05  ADTBUFFL         PIC  9(8) COMP VALUE 0.
         05  ADTRTNCD         PIC  9(8) COMP VALUE 0.
             88  ADT-RCOKAY              VALUE  0.
             88  ADT-NO-MORE-DATA        VALUE 65.
             88  ADT-WARNING            VALUE  1 THRU 15.
             88  ADT-ERROR              VALUE 16 THRU 255.
         05  ADTDGNCD         PIC  9(8) COMP VALUE 0.
         05  ADTSTATS         PIC  9(8) COMP VALUE 0.
         05  ADTTRACE         PIC  9(8) COMP VALUE 0.
         05  ADTQSEND         PIC  9(8) COMP VALUE 0.
         05  ADTMSEND         PIC  9(8) COMP VALUE 0.
         05  ADTQRECV         PIC  9(8) COMP VALUE 0.
         05  ADTMRECV         PIC  9(8) COMP VALUE 0.
         05  ADTTIMEO         PIC  9(8) COMP VALUE 0.
         05  FILLER           PIC  9(8) COMP VALUE 0.
         05  FILLER           PIC  9(8) COMP VALUE 0.
         05  FILLER           PIC  9(8) COMP VALUE 0.
         05  FILLER           PIC  9(8) COMP VALUE 0.
         05  ADTLPORT         PIC  9(4) COMP VALUE 0.
         05  ADTRPORT         PIC  9(4) COMP VALUE 0.
```

```
05  FILLER          PIC   9(4) COMP VALUE 0.
05  ADTSRVCE        PIC  X(36) VALUE ' '.
05  ADTSEPN         PIC   9(4) COMP VALUE 0.
    88 ADTSEPN-1                VALUE 1.
    88 ADTSEPN-2                VALUE 2.
05  ADTSEP1         PIC   X(1) VALUE ' '.
05  ADTSEP2         PIC   X(1) VALUE ' '.
05  FILLER          PIC   9(4) COMP VALUE 0.
05  ADTLADDR        PIC   9(8) COMP VALUE 0.
05  ADTRADDR        PIC   9(8) COMP VALUE 0.
05  ADTLNAME        PIC X(255) VALUE ' '.
05  FILLER          PIC   X(1) VALUE LOW-VALUE.
05  ADTRNAME        PIC X(255) VALUE ' '.
05  FILLER          PIC   X(1) VALUE LOW-VALUE.
05  ADTUCNTX        PIC   9(8) COMP VALUE 0.
05  ADTOPTNS        PIC   9(8) COMP VALUE 0.
```

The following constants are contained in the T09MAC library member, T09KCCON:

```
01  ADT-CONSTANTS.

*---------------------------------------------------------------*
*  ADTSTATS AND ADTTRACE VALUES ARE THE SAME AS DEFINED FOR     *
*  THE ACM, AND ARE SET ONLY FOR UDP CALLS, RCVFROM AND SENDTO*
*  TO SET THESE FIELDS, USE THE EQUIVALENT ACM-CONSTANT.        *
*---------------------------------------------------------------*
*   ADTOPTNS        SPECIAL SEND AND RECEIVE PROCESSING MODES.*
*---------------------------------------------------------------*
     * SEPARATOR CHARACTER SEND/RECEIVE
        05  ADTOPTNS-TYPSP    PIC   9(8) COMP VALUE     1.
     * LENGTH (LL) SEND/RECEIVE
        05  ADTOPTNS-TYPLL    PIC   9(8) COMP VALUE     2.
     * BLOCK ON SEND
        05  ADTOPTNS-BLCKS    PIC   9(8) COMP VALUE     4.
     * TIMED PARTIAL RECEIVE
        05  ADTOPTNS-TMPRT    PIC   9(8) COMP VALUE     8.
     * TIMED FULL BLOCK RECEIVE
        05  ADTOPTNS-TMRCV    PIC   9(8) COMP VALUE    16.
     * DO NOT BLOCK ON RECEIVE/RCVFROM
        05  ADTOPTNS-NBLKR    PIC   9(8) COMP VALUE    64.
     * DO DNR NAME RESOULTION IN UDP
        05  ADTOPTNS-DODNR    PIC   9(8) COMP VALUE   128.
    * LEAVE SEPARATOR OR LENGTH BYTES WITH DATA
        05  ADTOPTNS-NOSTP    PIC   9(8) COMP VALUE 16384.
     * VECTOR LIST FLAG
        05  ADTOPTNS-FVLST    PIC   9(8) COMP VALUE 32768.
```

PARAMETER	DESCRIPTION
ADTVERS	Version Indicates the CPT version number of the argument used by the calling program. This required field is set to 2 for this release of CPT. Default: 2

200801-024040-200100

PARAMETER	DESCRIPTION
ADTFUNC	Function code Indicates the function or callable service ID requested by the application program. This field is not set by the application, but is initialized by the TRUE interface stub program. Default: None
ADTTOKEN	Data transfer token Specifies a token that represents a TCP connection. If the ADT is being passed in a call to either the RECEIVE or SEND service, then it must be a token representing a previously established TCP connection, via the CONNECT or LISTEN service. it is an error to pass a zero ADTTOKEN to either the RECEIVE or SEND service. It is an error to pass a TCP token to the UDP data transfer service routines, RCVFROM and SENDTO. Conversely, it is an error to pass a UDP token to the TCP data transfer routines. RECEIVE and SEND. Default: 0
ADTBUFFA	User data address Indicates the storage address into which network data is placed. This is a contiguous segment of storage accessible to the user task. The storage area can be aligned on any boundary convenient for the application program. Default: 0
ADTINCRA	Redefines ADTBUFFA Allows for computational updating of the user data address.
ADTBUFFL	User data length Indicates the length (in bytes) of user data in the storage area as identified by the ADTBUFFA operand. The length is updated when the request is completed to reflect the actual length of user data received. This field must be interpreted on completion to determine the amount of data actually received. If a RECEIVE request is issued with a zero length, an error is detected and the request fails. Default: 0
ADTRTNCD	Return code Indicates the return code set by the RECEIVE service. This value is returned and indicates the success or failure of the service. Default: 0
ADTDGNCD	Diagnostic code Indicates the diagnostic code set by the service request. This value generally indicates a transport provider return code. Default: 0
ADTSTATS	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
ADTTRACE	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
ADTQSEND	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
ADTMSEND	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.

2024040-200100

PARAMETER	DESCRIPTION
ADTQRECV	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
ADTMRECV	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
ADTTIMEO	RECEIVE timeout value Must be specified with these options: ◆ ADTOPTNS-TYPLL ◆ ADTOPTNS-TYPSP ◆ ADTOPTNS-TMRCV ◆ ADTOPTNS-TMPRT Specifying any of the above options on a RECEIVE call with an ADTTIMEO=ZERO will result in CPTETIME being returned in ADTRTNCD. Default: None
ADTLPORT	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
ADTRPORT	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
ADTSRVCE	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
ADTSEPN	Number of separator characters for option ADTOPTNS-TYPSP (0 < ADTSEPN< 3) If ADTSEPN is not equal to one or two, CPTESEP# will be returned in ADTRTNCD. Default: None
ADTSEPN-1	First or only separator character for option ADTOPTNS-TYPSP. Default: None
ADTSEPN-2	Second separator character in a sequence of two for option ADTOPTNS-TYPSP. Default: None
ADTLADDR	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
ADTRADDR	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
ADTLNAME	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
ADTRNAME	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
ADTUCNTX	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.

200801-024040-200100

PARAMETER	DESCRIPTION
ADTOPTNS	This field specifies data transfer options. These are the ADT options that apply to TCP data transfer requests: ◆ ADTOPTNS-NBLKR – Do not block on a call to the RECEIVE service If no data is currently available on the connection, CPTWBLCK will be returned in ADTRTNCD. ◆ ADTOPTNS-TMRCV – Timed full record RECEIVE These fields (along with other required ADT fields) are used to request a timed full record RECEIVE: ADTOPTNS-TMRCV ADTTIMEO > zero ADTBUFFL set to the length expected If the time limit expires before receiving any or all of the data specified by ADTBUFFL, CPTWTIMO will be returned in ADTRTNCD along with any data that was received. ◆ ADTOPTNS-TMPRT – Timed partial record RECEIVE These fields (along with other required ADT fields) are used to request a timed partial record RECEIVE: ADTOPTNS-TMPRT ADTTIMEO > zero ADTBUFFL set to maximum length expected If the time limit expires before receiving any data, CPTWTIMO will be returned in ADTRTNCD. If the time limit expires and any data is received, the data, along with a zero ADTRTNCD, will be returned to the caller. ◆ ADTOPTNS-TYPSP – SEP type RECEIVE These fields (along with other required ADT fields) are used to request a SEP type RECEIVE call: ADTOPTNS-TYPSP ADTSEPN = 1 or 2 ADTSEPN-1= character ADTSEPN-2= character if ADTSEPN = 2 ADTTIMEO > zero If the time limit expires before receiving data, CPTWTIMO is returned in ADTRTNCD. If the time limit expires and any data is received, the data, along with a zero ADTRTNCD, will be returned to the caller. ◆ ADTOPTNS-TYPLL – LL type RECEIVE These fields (along with other required ADT fields) are used to request a SEP type RECEIVE call: ADTOPTNS-TYPLL ADTTIMEO > zero If the time limit expires before receiving any or all of the data specified by the LL (first two bytes of the data stream), CPTWTIMO will be returned in ADTRTNCD, along with any data that was received. ◆ ADTOPTNS-NOSTP – Do not strip record delimiter sequence This can be used with ADTTYPSP or ADTTYPLL to return the actual separator sequence or LL field in the buffer pointed to by ADTBUFFA. ◆ ADTOPTNS-BLCKS – Block on SEND service call (not used by RECEIVE service). ◆ ADTOPTNS-FVLST – Currently internal use only **Note:** It is an error to combine any of these RECEIVE service options: ADTOPTNS-NBLKR, ADTOPTNS-TYPLL, ADTOPTNS-TYPSP, ADTOPTNS-TMRCV, ADTOPTNS-TMPRT. An invalid combination causes CPTEOPTN to return in ADTRTNCD. Default: None

Completion Information

The RECEIVE service completes normally when the data is moved from the transport provider buffer to the application program's storage area. A length is returned to the application program, which is set to the amount of data actually processed.

Normal completion of the RECEIVE service implies that data has been moved to the user buffer. This does not necessarily indicate the application request was completely satisfied, but that some amount of data was processed. The user application is required to load the ADTBUFFL field to determine the actual data received. The RECEIVE service returns control to the calling application on receipt of a full buffer, a partial buffer, or an error indication. Control is returned to the user application with a partial buffer to avoid a WAIT command within the RECEIVE service. Additional requests to the RECEIVE service may be required to completely satisfy the user application's requirement.

The presence of exceptions or error conditions do not always indicate serious errors. A user application should check the return code to determine proper flow control. The release indication return code is an example of a condition that is not necessarily a serious error. This exception specifies that the remote host has closed it's half of the full-duplex data connection and will not send any additional data. This return code is acceptable, and generally indicates that graceful termination of the connection should begin.

On normal return to the application program, the general return code in ADTRTNCD is set to zero (CPTIRCOK). The diagnostic code in ADTDGNCD is always zero. The length field (ADTBUFFL) indicates the amount of data processed.

If the RECEIVE service completes abnormally, some or no user data may have been sent to the peer transport user. The general return code in ADTRTNCD, and the diagnostic code in ADTDGNCD, indicate the nature of the failure. The diagnostic code (ADTDGNCD) generally contains a specific code that is generated by the transport provider.

200801-024040-200100

Return Codes

The RECEIVE service return and diagnostic codes indicate the result of the execution. These values are in the ADTRTNCD and ADTDGNCD within the ADT. The diagnostic code is optional and generally indicates a transport provider return code. Read **Appendix C – MESSAGES AND CODES** in the ***CICS Programmer's Toolkit Installation and Administration Guide*** for the return code cross reference table.

This table describes the RECEIVE service return codes:

RETURN	DIAGNOSTIC CODE	DESCRIPTION
CPTIRCOK		Successful.
CPTEVERS		Control block version number is not supported.
CPTETOKN		Specified token is not valid.
CPTEBUFF		Buffer address and/or length is invalid.
CPTENAPI	Yes	Transport provider API is not available.
CPTETERM	Yes	Environment is being terminated.
CPTERLSE	Yes	Release indication.
CPTEDISC	Yes	Disconnect indication.
CPTEINTG	Yes	Transport provider API integrity error.
CPTEENVR	Yes	Transport provider API environment error.
CPTEFRMT	Yes	Transport provider API Format error.
CPTEPROC	Yes	Transport provider API procedure error.
CPTABEND		Abnormal exception occurred.
CPTEOTHR	Yes	An undefined exception occurred.

Usage Information

The RECEIVE service receives normal data as input through a CPT connection. The data is part of a byte stream being received over a connection (TCP).

Data is moved from the transport provider's storage area to the user application's storage area. Stream data may not be received with the same logical boundaries with which it was sent. However, the data arrives in the precise order in which it was sent. Possible fragmentation is a characteristic of stream data.

User applications may be required to issue multiple RECEIVE service requests to obtain all of the desired data. The data may arrive in particle segments. An application should be designed to handle such a situation. Additionally, users who write applications to process multiple record oriented data should consider including a mechanism to delimit the data. Design options can include

20-024040-200100

a logical length field at the beginning of a record, or a special field or fields at the end. This lets the application determine record boundaries.

The RECEIVE service request is a synchronous operation that may require the application to be blocked. The RECEIVE queue (ACMQRECV) and buffer (ACMMRECV) sizes, as specified during connection initialization, control input queuing and buffering. The queue size represents the number of uncompleted RECEIVE requests that can be pending to the transport provider. The default value is generally sufficient to support most applications. The transport provider normally completes a RECEIVE request on transferring data into the user buffer. However, data over the network may be fragmented and may require multiple requests to retrieve all of the data. The RECEIVE service issues a WAIT command if no data is available.

The queue and buffer size values are specified during connection initialization and can be modified by either the LISTEN or CONNECT services. An application that is dependent on these values should validate the requested values compared with those values returned within the ACM. The values are modified if the transport provider site administrator has configured limits and the application request exceeds those values. If the requested values are modified, verify site definition statements for API transport services.

The version number (ADTVERS) indicates the CPT release level in which this user application program is written. This required field must be set to ADTVERS2 and is validated by the RECEIVE service before processing the request.

The function code (ADTFUNC) indicates the CPT callable service ID. The field is not initialized by a user application program and has little value to the application except for dump analysis. The function code identifies and maps an argument list with the error or trace log and dump analysis.

The token (ADTTOKEN) indicates the connection that is to receive data. This is a required field and is validated by the RECEIVE service before it processes the request.

The data buffer address field ADTBUFFA is a full word. The application program assures that the residency mode of data areas it manages (for example, argument lists) is compatible with the addressing mode. The transport provider performs consistency checks on the addressing mode whenever a service request is issued. However, unpredictable results may occur before the transport provider can perform this check.

The data buffer length is indicated by the ADTBUFFL field. This is a full word positive integer. The data buffer length field should be less than or equal to the maximum receive buffer values. However, if the data buffer length is greater than the maximum receive buffer, the RECEIVE service attempts to satisfy the user's request with multiple transport provider requests. On return from the RECEIVE service, the ADTBUFFL is updated with a value that indicates the number of bytes processed.

The ADTOPTNS field specifies RECEIVE processing control options and provides a mechanism for event notification on return to the application program.

200801-024040-200100

Example:

This example establishes a connection and the application receives data from the remote host. The token is loaded from the ACM and used by all of the following CPT service requests. The return code is checked to determine RECEIVE service completion status.

```
WORKING-STORAGE SECTION.
01  CPT-ACM    COPY T09KCACM.
01  CPT-ADT    COPY T09KCADT.
01  CPT-ACL    COPY T09KCACL.
01  CPT-RCS    COPY T09KCRCS.

LINKAGE SECTION.
01   MESSAGEPIC X(1023).

PROCEDURE DIVISION.

*
* CPT Connection Management initialization and request
*
     .
     .
     .
    MOVE ACMTOKEN TO ADTTOKEN ACLTOKEN.

*
* Application and CPT RECEIVE Data Transfer processing
*
    PERFORM UNTIL NOT ADT-RCOKAY
        SET ADTBUFFA TO ADDRESS OF MESSAGE
        MOVE 80 TO ADTBUFFL
        CALL 'T09FRECV' USING CPT-ADT
        IF NOT ADT-RCOKAY
*           Process and log RECEIVE service error
        END-IF
    END-PERFORM.

    IF ADT-NO-MORE-DATA
*
*       CPT Orderly Connection Release processing
*
    ELSE
*
*       CPT Abortive Connection Release processing
*
    END-IF.

*
* Terminate Transaction
*
    EXEC CICS RETURN
    END-EXEC.
```


2024040-200100

SEND

The SEND service sends data to a peer transport user connected to an endpoint. The SEND service sends data as output on a connection-mode (TCP) endpoint only.

To invoke the SEND service, a user application is required to first build an ADT and then to issue a call to the SEND routine. The ADT is required to contain the version number, connection token, user buffer address, and length. On completion of the SEND service, the buffer length field is updated to reflect the amount of data processed.

This table describes the arguments for the SEND service:

COPY NAME	SIZE	CREATED BY
T09KCADT	644 (X'284')	User application

Structure

This is the structure of the SEND service in COBOL language:

```
*  01   CPT-ADT.
                  COPY T09KCADT.

*****************************************************************
*                          CPT-ADT                              *
*     CICS PROGRAMMER'S TOOLKIT DATA TRANSFER CONTROL BLOCK     *
*---------------------------------------------------------------*
          05  ADTVERS           PIC   9(4) COMP VALUE 2.
              88  ADTVERSN     VALUE 2.
          05  ADTFUNC           PIC   9(4) COMP VALUE 0.
          05  ADTTOKEN          PIC   9(8) COMP VALUE 0.
          05  ADTBUFFA          POINTER    VALUE NULL.
         05  ADTINCRA REDEFINES ADTBUFFA PIC 9(8) COMP.
          05  ADTBUFFL          PIC   9(8) COMP VALUE 0.
          05  ADTRTNCD          PIC   9(8) COMP VALUE 0.
              88  ADT-RCOKAY               VALUE  0.
              88  ADT-NO-MORE-DATA         VALUE 65.
              88  ADT-WARNING             VALUE  1 THRU 15.
              88  ADT-ERROR               VALUE 16 THRU 255.
          05  ADTDGNCD          PIC   9(8) COMP VALUE 0.
          05  ADTSTATS          PIC   9(8) COMP VALUE 0.
          05  ADTTRACE          PIC   9(8) COMP VALUE 0.
          05  ADTQSEND          PIC   9(8) COMP VALUE 0.
          05  ADTMSEND          PIC   9(8) COMP VALUE 0.
          05  ADTQRECV          PIC   9(8) COMP VALUE 0.
          05  ADTMRECV          PIC   9(8) COMP VALUE 0.
          05  ADTTIMEO          PIC   9(8) COMP VALUE 0.
          05  FILLER            PIC   9(8) COMP VALUE 0.
          05  FILLER            PIC   9(8) COMP VALUE 0.
          05  FILLER            PIC   9(8) COMP VALUE 0.
          05  FILLER            PIC   9(8) COMP VALUE 0.
          05  ADTLPORT          PIC   9(4) COMP VALUE 0.
          05  ADTRPORT          PIC   9(4) COMP VALUE 0.
```

```
05  FILLER              PIC   9(4) COMP VALUE 0.
05  ADTSRVCE            PIC  X(36) VALUE ' '.
05  ADTSEPN             PIC   9(4) COMP VALUE 0.
    88 ADTSEPN-1                   VALUE 1.
    88 ADTSEPN-2                   VALUE 2.
05  ADTSEP1             PIC   X(1) VALUE ' '.
05  ADTSEP2             PIC   X(1) VALUE ' '.
05  FILLER              PIC   9(4) COMP VALUE 0.
05  ADTLADDR            PIC   9(8) COMP VALUE 0.
05  ADTRADDR            PIC   9(8) COMP VALUE 0.
05  ADTLNAME            PIC X(255) VALUE ' '.
05  FILLER              PIC   X(1) VALUE LOW-VALUE.
05  ADTRNAME            PIC X(255) VALUE ' '.
05  FILLER              PIC   X(1) VALUE LOW-VALUE.
05  ADTUCNTX            PIC   9(8) COMP VALUE 0.
05  ADTOPTNS            PIC   9(8) COMP VALUE 0.
```

The following constants are contained in the T09MAC library member, T09KCCON:

```
01  ADT-CONSTANTS.

*-------------------------------------------------------------*
*  ADTSTATS AND ADTTRACE VALUES ARE THE SAME AS DEFINED FOR   *
*  THE ACM, AND ARE SET ONLY FOR UDP CALLS, RCVFROM AND SENDTO*
*  TO SET THESE FIELDS, USE THE EQUIVALENT ACM-CONSTANT.      *
*-------------------------------------------------------------*
*   ADTOPTNS        SPECIAL SEND AND RECEIVE PROCESSING MODES.*
*-------------------------------------------------------------*
     * SEPARATOR CHARACTER SEND/RECEIVE
        05  ADTOPTNS-TYPSP    PIC  9(8) COMP VALUE    1.
     * LENGTH (LL) SEND/RECEIVE
        05  ADTOPTNS-TYPLL    PIC  9(8) COMP VALUE    2.
     * BLOCK ON SEND
        05  ADTOPTNS-BLCKS    PIC  9(8) COMP VALUE    4.
     * TIMED PARTIAL RECEIVE
        05  ADTOPTNS-TMPRT    PIC  9(8) COMP VALUE    8.
     * TIMED FULL BLOCK RECEIVE
        05  ADTOPTNS-TMRCV    PIC  9(8) COMP VALUE   16.
     * DO NOT BLOCK ON RECEIVE/RCVFROM
        05  ADTOPTNS-NBLKR    PIC  9(8) COMP VALUE   64.
     * DO DNR NAME RESOULTION IN UDP
        05  ADTOPTNS-DODNR    PIC  9(8) COMP VALUE  128.
    * LEAVE SEPARATOR OR LENGTH BYTES WITH DATA
        05  ADTOPTNS-NOSTP    PIC  9(8) COMP VALUE 16384.
     * VECTOR LIST FLAG
        05  ADTOPTNS-FVLST    PIC  9(8) COMP VALUE 32768.
```

PARAMETER	DESCRIPTION
ADTVERS	Version Indicates the CPT version number of the argument used by the calling program. This required field is set to 2 for this release of CPT. Default: 2

PARAMETER	DESCRIPTION
ADTFUNC	Function code Indicates the function or callable service ID requested by the application program. This field is not set by the application, but is initialized by the TRUE interface stub program. Default: None
ADTTOKEN	Data transfer token Specifies a token that represents a TCP connection. If the ADT is being passed in a call to either the RECEIVE or SEND service, then it must be a token representing a previously established TCP connection, via the CONNECT or LISTEN service. it is an error to pass a zero ADTTOKEN to either the RECEIVE or SEND service. It is an error to pass a TCP token to the UDP data transfer service routines, RCVFROM and SENDTO. Conversely, it is an error to pass a UDP token to the TCP data transfer routines. RECEIVE and SEND. Default: 0
ADTBUFFA	User data address Indicates the storage address into which network data is placed. This is a contiguous segment of storage accessible to the user task. The storage area can be aligned on any boundary convenient for the application program. Default: 0
ADTINCRA	Redefines ADTBUFFA Allows for computational updating of the user data address.
ADTBUFFL	User data length Indicates the length (in bytes) of user data in the storage area as identified by the ADTBUFFA operand. The length is updated when the request is completed to reflect the actual length of user data received. This field must be interpreted on completion to determine the amount of data actually received. If a RECEIVE request is issued with a zero length, an error is detected and the request fails. Default: 0
ADTRTNCD	Return code Indicates the return code set by the SEND service. This value is returned and indicates the success or failure of the service. Default: 0
ADTDGNCD	Diagnostic code Indicates the diagnostic code set by the service request. This value generally indicates a transport provider return code. Default: 0
ADTSTATS	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
ADTTRACE	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
ADTQSEND	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
ADTMSEND	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.

200801-024040-200100

PARAMETER	DESCRIPTION
ADTQRECV	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
ADTMRECV	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
ADTTIMEO	RECEIVE timeout value Not used by the SEND service. Default: None
ADTLPORT	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
ADTRPORT	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
ADTSRVCE	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
ADTSEPN	Number of separator characters for option ADTOPTNS-TYPSP (0 < ADTSEPN < 3) If ADTSEPN is not equal to one or two, CPTESEP# will be returned in ADTRTNCD. Default: None
ADTSEPN-1	First or only separator character for option ADTOPTNS-TYPSP. Default: None
ADTSEPN-2	Second separator character in a sequence of two for option ADTOPTNS-TYPSP. Default: None
ADTLADDR	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
ADTRADDR	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
ADTLNAME	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
ADTRNAME	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
ADTUCNTX	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.

2024040-200100

PARAMETER	DESCRIPTION

ADTOPTNS	This field specifies data transfer options.
	These are the ADT options that apply to TCP data transfer requests:
	◆ ADTOPTNS-NBLKR – Do not block on a call to the RECEIVE service. Not used by the SEND service.
	◆ ADTOPTNS-TMRCV – Timed full record RECEIVE. Not used by the SEND service.
	◆ ADTOPTNS-TMPRT – Timed partial record RECEIVE. Not used by the SEND service.
	◆ ADTOPTNS-TYPSP – SEP type SEND. These files (along with other required ADT fields) are used to request a SEP type SEND call:
	`ADTOPTNS = ADTOPTN1-TYPSP`
	`ADTSEPN = 1 OR 2`
	`ADTSEP1 = character`
	`ADTSEP2 = character if ADTSEP# = 2`
	◆ ADTOPTNS-TYPLL – LL type SEND. These fields (along with other required ADT fields) are used to request a SEP type SEND call:
	`ADTOPTNS = ADTOPTNS-TYPLL`
	◆ ADTOPTNS-BLCKS – Block on SEND service call. The default for the SEND service is to return control to the caller as soon as the SEND request has been scheduled with the local transport provider. Option ADTOPTNS-BLCKS will block the return to the caller until the SEND data has been moved to the local transport provider's address space (not the remote TCP). This can provide TCP connection status at the SEND service call time rather than at some later time (next SEND or RECEIVE call).
	Note: It is an error to combine any of these SEND service options:
	`ADTOPTNS-TYPLL`
	`ADTOPTNS-TYPSP`
	An invalid combination will result in CPTEOPTN being returned in ADTRTNCD.
	Default: None

200801-024040-200100

Completion Information

The SEND service completes normally when the data is both moved from the storage area of the application program and forwarded to the transport provider for sending to the connected transport user. A length is returned to the application program which is set to the amount of data processed.

Normal completion of the SEND service implies nothing in regard to when the data is sent to the peer transport user. This only means that the transport provider has taken custody of the user data and the storage area provided by the application program can be reused. The transport provider generally sends the buffered data, but this may not occur synchronously with the completion of the SEND service.

On normal return to the application program, the general return code in ADTRTNCD is set to zero (CPTIRCOK). The diagnostic code in ADTDGNCD is always zero. The length field (ADTBUFFL) indicates amount of data processed.

If the SEND service completes abnormally, some or no user data may have been sent to the peer transport user. The general return code in ADTRTNCD and the diagnostic code in ADTDGNCD, indicate the nature of the failure. The diagnostic code (ADTDGNCD) generally contains a specific code that is generated by the transport provider.

Return Codes

The SEND service return and diagnostic codes indicate the result of the execution. These values are in the ADTRTNCD and ADTDGNCD within the ADT. The diagnostic code is optional and generally indicates a transport provider return code. Read **Appendix C – MESSAGES AND CODES** in the ***CICS Programmer's Toolkit Installation and Administration Guide*** for the return code cross reference table.

This table describes the SEND service return codes:

RETURN	DIAGNOSTIC CODE	DESCRIPTION
CPTIRCOK		Successful.
CPTEVERS		Control block version number is not supported
CPTETOKN		Specified token is not valid
CPTEBUFF		Buffer address and/or length is invalid.
CPTEPRGE	Yes	CPT Interface is terminating
CPTENAPI	Yes	Transport provider API is not available.
CPTETERM	Yes	Environment is being terminated.
CPTERLSE	Yes	Release indication.
CPTEDISC	Yes	Disconnect indication.
CPTEINTG	Yes	Transport provider API integrity error.

20-024040-200100

RETURN	DIAGNOSTIC CODE	DESCRIPTION
CPTEENVR	Yes	Transport provider API environment error.
CPTEFRMT	Yes	Transport provider API format error.
CPTEPROC	Yes	Transport provider API procedure error.
CPTABEND		Abnormal exception occurred.
CPTEOTHR	Yes	An undefined exception occurred.

Usage Information

The SEND service sends normal data as output through a CPT connection. The data is part of a byte stream being sent over a connection (TCP).

Data is moved from the storage area of the application program to storage areas maintained by the transport provider. It is packetized and sent to the connection transport user. Logical boundaries are not preserved in the data stream. The data is delivered to the peer transport user in the precise order in which it was sent. However, this data may be fragmented.

Data is not necessarily packetized and sent by the transport provider each time a SEND service is issued, nor is it sent when a buffer boundary is indicated. The transport provider may intentionally delay sending data as the result of performance optimization or congestion avoidance algorithms. Normally, data generated by the application is forwarded when it is sent in a continuous flow.

The SEND service request is a synchronous operation that may require the application to be blocked. The SEND queue (ACMQSEND) and buffer (ACMMSEND) sizes, as specified during connection initialization, control output queuing and buffering. The queue size represents the number of uncompleted SEND requests that can be pending to the transport provider. The default value is generally sufficient to support most applications. The transport provider normally completes a SEND request within a reasonable amount of time, but a congested network may cause the transport provider delays in completing requests, and cause the application to be blocked.

Additionally, for an application that issues multiple SEND requests contiguously, consider increasing the default queue size. The buffer size represents the maximum number of user data bytes that can be transferred by the application in a single SEND request to the transport provider. This value is application dependent. A small value causes the SEND service to issue multiple transport provider SEND requests. Multiple transport provider SEND requests do not present a problem, but do reduce the queue size and can cause the application to be blocked. A large buffer value can waste application storage.

The queue and buffer size values are specified during connection initialization and can be modified on return. An application that is dependent on these values should validate the requested values compared with values returned within the ACM. The values are modified if the transport provider site administrator has configured limits and the application request exceeds those values. If the requested values are modified, verify site definition statements for API transport services.

200801-024040-200100

The version number (ADTVERS) indicates the CPT release level in which this user application program is written. This required field must be set to ADTVERS2 and is validated by the SEND service before processing the request.

The function code (ADTFUNC) indicates the CPT callable service ID. The field is not initialized by a user application program and has little value to the application except for dump analysis. The function code identifies and maps an argument list with the error or trace log and dump analysis.

The token (ADTTOKEN) indicates the connection that will transmit data. This required field is validated by the SEND service before processing the request.

The data buffer address field ADTBUFFA is a full word. The application program assures that the residency mode of data areas it manages (for example, argument lists) is compatible with the addressing mode. The transport provider performs consistency checks on the addressing mode whenever a service request is issued. However, unpredictable results may occur before the transport provider performs this check.

The data buffer length is indicated by the ADTBUFFL field. This is a full word positive integer. The data buffer length field should be less than or equal to the maximum send buffer values. however, if the data buffer length is greater than the maximum send buffer, the SEND service fragments the user data into multiple transport provider requests. The ADTBUFFL is updated on return from the SEND service with a value that indicates the number of bytes processed.

The ADTOPTNS field specifies SEND processing control options and provides a mechanism for event notification on return to the application program.

2024040-200100

Example:

In the example, a connection has been established and a welcome message is sent to the remote host. The token is loaded from the ACM and used by all of the following CPT service requests. The return code is checked to determine SEND service completion status.

```
 WORKING-STORAGE SECTION.
 01 CPT-ACM    COPY T09KCACM.
 01 CPT-ADT    COPY T09KCADT.
 01 CPT-ACL    COPY T09KCACL.
 01 CPT-RCS    COPY T09KCRCS.

 LINKAGE SECTION.
 01 MESSAGE    PIC X(1024)
          VALUE 'WELCOME TO A CICS TEST APPLICATION'.

 PROCEDURE DIVISION.

*
* CPT Connection Management initialization and request
*
    .
    .
    .
    MOVE ACMTOKEN TO ADTTOKEN ACLTOKEN.

*
* Application and CPT Send Data Transfer processing
*
    SET ADTBUFFA TO ADDRESS OF MESSAGE.
    These messages are 80 TO ADTBUFFL.
    CALL 'T09FSEND' USING CPT-ADT.
    IF NOT ADT-RCOKAY
*       Process and log SEND service error
    END-IF.

*   CPT Orderly Connection Release processing
*

*
* Terminate Transaction
*
    EXEC CICS RETURN
    END-EXEC.
```

200801-024040-200100

SENDTO

This service is provided to allow connectionless client and server applications to be developed. This service is UDP only. The SENDTO service provides two basic functions:

- Establish a UDP client endpoint represented by a new token and send a datagram to a remote UDP server. This function is indicated to the SENDTO service by passing an ADTTOKEN equal to zero. SENDTO will then create all the internal control blocks and the SENDTO buffer queue. Even though the RCVFROM buffer queue will not be allocated for this endpoint (token) until the RCVFROM service is called, the RCVFROM buffer size and number must be specified at this time because they are negotiated with the transport provider and recorded in the internal CPT control blocks at endpoint creation time. Upon return from the SENDTO service ADTTOKEN will contain the token value to be passed to subsequent SENDTO and RCVFROM service calls.
- Send a datagram at a previously established UDP endpoint represented by an existing token. This functionality makes the SENDTO service call just a data transfer call that can be used by a client or server application. The SENDTO buffer queue is only allocated upon the first call to the SENDTO service whether ADTTOKEN is equal to zero or not.

UDP tokens created with the RCVFROM or SENDTO services cannot be passed to the TCP only services, CONNECT, LISTEN, SEND, and RECEIVE. All other CPT service calls are available to UDP applications.

This table describes the arguments for the SEND service:

COPY NAME	SIZE	CREATED BY
T09KCADT	644 (X'284')	User application

Structure

This is the structure of the SENDTO service in COBOL language:

```
*  01  CPT-ADT.
                COPY T09KCADT.

*****************************************************************
*                          CPT-ADT                              *
*     CICS PROGRAMMER'S TOOLKIT DATA TRANSFER CONTROL BLOCK     *
*---------------------------------------------------------------*
          05  ADTVERS          PIC   9(4) COMP VALUE 2.
              88  ADTVERSN     VALUE 2.
          05  ADTFUNC          PIC   9(4) COMP VALUE 0.
          05  ADTTOKEN         PIC   9(8) COMP VALUE 0.
          05  ADTBUFFA         POINTER    VALUE NULL.
         05  ADTINCRA REDEFINES ADTBUFFA PIC 9(8) COMP.
          05  ADTBUFFL         PIC   9(8) COMP VALUE 0.
          05  ADTRTNCD         PIC   9(8) COMP VALUE 0.
              88  ADT-RCOKAY              VALUE  0.
```

```
           88  ADT-NO-MORE-DATA        VALUE 65.
           88  ADT-WARNING            VALUE  1 THRU 15.
           88  ADT-ERROR              VALUE 16 THRU 255.
       05  ADTDGNCD         PIC   9(8) COMP VALUE 0.
       05  ADTSTATS         PIC   9(8) COMP VALUE 0.
       05  ADTTRACE         PIC   9(8) COMP VALUE 0.
       05  ADTQSEND         PIC   9(8) COMP VALUE 0.
       05  ADTMSEND         PIC   9(8) COMP VALUE 0.
       05  ADTQRECV         PIC   9(8) COMP VALUE 0.
       05  ADTMRECV         PIC   9(8) COMP VALUE 0.
       05  ADTTIMEO         PIC   9(8) COMP VALUE 0.
       05  FILLER           PIC   9(8) COMP VALUE 0.
       05  FILLER           PIC   9(8) COMP VALUE 0.
       05  FILLER           PIC   9(8) COMP VALUE 0.
       05  FILLER           PIC   9(8) COMP VALUE 0.
       05  ADTLPORT         PIC   9(4) COMP VALUE 0.
       05  ADTRPORT         PIC   9(4) COMP VALUE 0.
       05  FILLER           PIC   9(4) COMP VALUE 0.
       05  ADTSRVCE         PIC  X(36) VALUE ' '.
       05  ADTSEPN          PIC   9(4) COMP VALUE 0.
           88 ADTSEPN-1                VALUE 1.
           88 ADTSEPN-2                VALUE 2.
       05  ADTSEP1          PIC   X(1) VALUE ' '.
       05  ADTSEP2          PIC   X(1) VALUE ' '.
       05  FILLER           PIC   9(4) COMP VALUE 0.
       05  ADTLADDR         PIC   9(8) COMP VALUE 0.
       05  ADTRADDR         PIC   9(8) COMP VALUE 0.
       05  ADTLNAME         PIC X(255) VALUE ' '.
       05  FILLER           PIC   X(1) VALUE LOW-VALUE.
       05  ADTRNAME         PIC X(255) VALUE ' '.
       05  FILLER           PIC   X(1) VALUE LOW-VALUE.
       05  ADTUCNTX         PIC   9(8) COMP VALUE 0.
       05  ADTOPTNS         PIC   9(8) COMP VALUE 0.
```

The following constants are contained in the `T09MAC` library member, `T09KCCON`:

```
01  ADT-CONSTANTS.

*---------------------------------------------------------------*
*  ADTSTATS AND ADTTRACE VALUES ARE THE SAME AS DEFINED FOR   *
*  THE ACM, AND ARE SET ONLY FOR UDP CALLS, RCVFROM AND SENDTO*
*  TO SET THESE FIELDS, USE THE EQUIVALENT ACM-CONSTANT.      *
*---------------------------------------------------------------*
*   ADTOPTNS       SPECIAL SEND AND RECEIVE PROCESSING MODES.*
*---------------------------------------------------------------*
     * SEPARATOR CHARACTER SEND/RECEIVE
         05  ADTOPTNS-TYPSP    PIC   9(8) COMP VALUE    1.
     * LENGTH (LL) SEND/RECEIVE
         05  ADTOPTNS-TYPLL    PIC   9(8) COMP VALUE    2.
     * BLOCK ON SEND
         05  ADTOPTNS-BLCKS    PIC   9(8) COMP VALUE    4.
     * TIMED PARTIAL RECEIVE
         05  ADTOPTNS-TMPRT    PIC   9(8) COMP VALUE    8.
     * TIMED FULL BLOCK RECEIVE
         05  ADTOPTNS-TMRCV    PIC   9(8) COMP VALUE   16.
     * DO NOT BLOCK ON RECEIVE/RCVFROM
         05  ADTOPTNS-NBLKR    PIC   9(8) COMP VALUE   64.
```

2008n1-024040-200100

```
* DO DNR NAME RESOULTION IN UDP
    05  ADTOPTNS-DODNR    PIC   9(8) COMP VALUE   128.
* LEAVE SEPARATOR OR LENGTH BYTES WITH DATA
    05  ADTOPTNS-NOSTP    PIC   9(8) COMP VALUE 16384.
* VECTOR LIST FLAG
    05  ADTOPTNS-FVLST    PIC   9(8) COMP VALUE 32768.
```

PARAMETER	DESCRIPTION
ADTVERS	Version Indicates the CPT version number of the argument used by the calling program. This required field is set to 2 for this release of CPT. Default: 2
ADTFUNC	Function code Indicates the function or callable service ID requested by the application program. This field is not set by the application, but is initialized by the TRUE interface stub program. Default: None
ADTTOKEN	Data transfer token Specifies a token that represents a UDP endpoint. If the ADT is being passed in a call to either the RCVFROM or SENDTO service, the token can be zero, indicating to either service, to first create a token before sending or receiving a datagram. If the token is not zero, it must be a token created previously by either the RCVFROM or SENDTO service. It is not necessary or efficient to create a token every time a CICS transaction calls the UDP data transfer services. It is an error to pass a TCP token to the UDP data transfer service routines, RCVFROM and SENDTO. Conversely, it is an error to pass a UDP token to the TCP data transfer routines, RECEIVE and SEND. Default: 0
ADTBUFFA	User data address Indicates the storage address from which the UDP datagram will be sent (SENDTO service). This is a contiguous segment of storage accessible to the user task. The content of all user data is application dependent, and not interpreted by either CPT nor the transport provider. The storage area can be aligned on any boundary convenient for the application program. Default: 0
ADTINCRA	Redefines ADTBUFFA Allows for computational updating of the user data address.
ADTBUFFL	User data length Indicates the length (in bytes) of the buffer specified in ADTBUFFA which is to be sent (SENDTO service). It is an error to call the SENDTO service with an ADTBUFFL of zero. Upon return to the caller, ADTBUFFL reflects the number of bytes actually sent (generally the number requested). Indicates the return code set by the SENDTO service. Default: 0
ADTRTNCD	Return code Indicates the return code set by the SENDTO service. This value is also returned in register 15 and indicates the success or failure of the service. Default: 0
ADTDGNCD	Diagnostic code Indicates the diagnostics code set by the SENDTO service. This value generally indicates a transport provider return code. Default: 0

PARAMETER	DESCRIPTION
ADTSTATS	Specifies statistics logging options for the application program. The facility can be used for debugging and tuning during development. ◆ ADTSTATS-CONN Specifies that a message(s) be generated on establishing either a listen service or a data transfer connection. These messages are generated by the CPT LISTEN and CONNECT services. The message numbers associated with this option are CPT802I and CPT803I. ◆ ADTSTATS-TERM Specifies that a message(s) is to be generated on termination of an established connection. These messages are generated by the CPT CLOSE service. The message numbers associated with this option are CPT807I, CPT808I, and CPT809I. Default: 0 (no statistics logging)
ADTTRACE	Specifies trace logging options for the application program. The facility can be used for debugging during development. ◆ ADTTRACE-NTRY – Specifies that a message be generated on entry to a CPT service routine. The message is generated by all CPT service routines. The message number associated with this option is CPT901I. ◆ ADTTRACE-ARGS – Specifies that a hex dump of the caller's arguments be generated on entry and exit of a CPT service routine. Messages are generated by the corresponding service routine. Message numbers associated with this option are CPT902I, CPT903I, CPT904I, CPT911I, CPT912, CPT921I, CPT922I, CPT924I, CPT925I, CPT930I, and CPT932I. ◆ ADTTRACE-RECV – Specifies that a hex dump of the transport provider's input (RECEIVE) data be logged. Messages are generated by the CPT RECV service. The message number associated with this option is CPT913I. ◆ ADTTRACE-SEND – Specifies that a hex dump of the transport provider's output (SEND) data be logged. Messages are generated by the CPT SEND service. The message number associated with this option is CPT914I. ◆ ADTTRACE-TERM – Specifies that a message be generated on termination of an CPT service routine. Messages are generated by all CPT service routines. The message number associated with this option is CPT908I. ◆ ADTTRACE-PASS – Specifies that a hex dump of resources related to a passed connection be logged. Messages are generated by the CPT GIVE and TAKE services. The message numbers associated with this option are CPT919I and CPT920I. ◆ ADTTRACE-CLSE – Specifies that a hex dump of resources related to a CLOSE processing of a connection be logged. Messages are generated by the CPT CLOSE service. The message number associated with this option is CPT906I ◆ ADTTRACE-TERR – Specifies that a hex dump of a transport provider API parameter list that fails successful completion be logged. The message number associated with this option is CPT400I. ◆ ADTTRACE-TOKN – Specifies that a hex dump of the connection token be logged. Messages are generated by all services routines on entry. The message number associated with this option is CPT909I. ◆ ADTTRACE-TPL – Specifies that a hex dump of the transport provider API parameter list be logged. The message numbers associated with this option are CPT917I and CTP935I. ◆ ADTTRACE-RLSE – Specifies that a message indicating a transport provider release indication be logged. Messages are generated by various CPT service routines. The message number associated with this option is CPT414I. ◆ ADTTRACE-STOR – Specifies that a hex dump of storage management argument be logged. Messages are generated by various CPT service routines. The message numbers associated with this option are CPT928I and CPT929I. Default: 0 (no trace logging)

200801-024040-200100

PARAMETER	DESCRIPTION
ADTQSEND	API send queue size (used when ADTTOKEN=0) Specifies the maximum number of uncompleted SENDTO requests that can be queued by the application to the transport provider (API). This value lets applications control output processing and can affect throughput rates. The value is negotiated with the transport provider and can be modified by the transport provider. The total allocation for output processing is the product of the SENDTO queue and buffer size values and cannot exceed 61K. Default: 4
ADTMSEND	API send buffer size (used when ADTTOKEN=0) Specifies the maximum number of user data bytes that can be transferred by the application in a single SENDTO request to the transport provider (API). This value lets applications control output processing and can affect throughput rates. The value is negotiated with the transport provider and can be modified by the transport provider. The total allocation for output processing is the product of the SENDTO queue and buffer size values and cannot exceed 61K. Default: 4096
ADTQRECV	API receive queue size (used when ADTTOKEN=0) Specifies maximum number of uncompleted RCVFROM requests that can be queued by the application to the transport provider (API). This value lets applications control input processing and can affect throughput rates. The value is negotiated with the transport provider and can be modified by the transport provider. The total allocation for input processing is the product of the RCVFROM queue and buffer size values and cannot exceed 61K. Default: 4
ADTMRECV	API RECEIVE buffer size (used when ADTTOKEN=0) Specifies the maximum number of user data bytes that can be transferred by the application in a single RCVFROM request to the transport provider (API). This value lets applications control input processing and can affect throughput rates. The value is negotiated with the transport provider and can be modified by the transport provider. The total allocation for input processing is the product of the RCVFROM queue and buffer size values and cannot exceed 61K. Default: 4096
ADTTIMEO	RECEIVE timeout value Not used by the SENDTO service Default: 0
ADTLPORT	Local well-known service port. Indicates the local transport layer port that the calling application will be sending (SENDTO) UDP datagrams from. If the SENDTO service creates the token, this port number is assigned by the transport layer and returned to the caller. If the RCVFROM service creates the token, this is the well-known port requested by the caller. This field is an unsigned positive integer with a maximum value of 65,534. The value must be unique for each server application. Default: None
ADTRPORT	Remote port. Indicates the remote transport layer port destination for the datagram being processed by the SENDTO service. This field is an unsigned positive integer with a maximum value of 65,534. Default: None
ADTSRVCE	Transport layer service name Not used in the SENDTO. Default: None
ADTSEPN	Number of separator characters for option ADTOPTNS-TYPSP. Not used in the SENDTO service. Default: None

PARAMETER	DESCRIPTION
ADTSEPN-1	First or only separator character for option ADTOPTNS-TYPSP Not used in the SENDTO service. Default: None
ADTSEPN-2	Second character or separator sequence for option ADTOPTNS-TYPSP. Not used in the SENDTO service. Default: None
ADTLADDR	Local IP host address Indicates the local host internet address. this field is an unsigned four-byte integer value. The local host internet address is returned to the caller of the SENDTO service. Default: None
ADTRADDR	Remote IP host address Indicates the remote host internet address destination for the datagram being processed by the SENDTO service. This field is a unsigned four-byte integer value. Default: None
ADTLNAME	Local IP host name Indicates the local host internet name. This field is a 255-byte character string that is padded with blanks. The local host internet name is returned to the caller of the SENDTO service. Default: None
ADTRNAME	Remote IP host name Indicates the remote host internet name. this field is a 255-byte character string that is padded with blanks. It is only resolved through internal DNR calls and returned to the caller of the UDP data transfer service routines (RCVFROM and SENDTO) if the ADTOPTNS flag, ADTOPTNS-DDNR, is specified. This is to prevent the DNR call overhead on every UDP data transfer call. Default: None
ADTUCNTX	One word of user context Specifies one arbitrary word of user context to be associated with the endpoint. The information provided is not interpreted by CPT, and is merely saved with other endpoint information. Default: 0 (No user context)
ADTOPTNS	Specifies data transfer options These are the ADT options that apply to UDP data transfer requests: ◆ ADTOPTNS-DODNR – Execute internal DNR calls during UDP data transfer service routine calls (RCVFROM and SENDTO) to resolve remote IP addresses into IP names in the ADTRNAME field. ◆ ADTOPTNS-NBLKR – Do not block on a call to the RCVFROM service (not used by the SENDTO service). Default: None These options can be toggled on every UDP data transfer call even if the caller is using the same token.

200801-024040-200100

Network Considerations

The ADT is a common data structure used for both client and server UDP applications. There are common and unique values specified for a particular service request.

This table describes network considerations for Assembler API:

NAME	SERVER CONDITIONS FOR RCVFROM	CLIENT CONDITIONS FOR SENDTO
ADTLPORT	Local server well-known port selected by user application.	Local assigned transport provider port returned to user application.
ADTRPORT	Remote client transport provider port returned to user by user application.	Remote server transport provider well-known port selected by user application.
ADTSRVCE	Local server trans port provider service name selected by user application.	Remote server transport provider service name selected by user application.
ADTRADDR	Remote IP host address returned to user application.	Remote IP host address selected by or returned to user application. The client must specify this field or ADTRNAME.
ADTLNAME	Local IP host name returned to user application.	Local IP host name returned to user application.
ADTRNAME	Remote IP host name returned to user application only if ADTOPTNS-DODNR is specified.	Remote IP host name selected by or returned to the user application. The client must specify this field or ADTRADDR. If ADTRADDR is used, ADTRNAME will only be returned if ADTOPTNS-DODNR is specified.

Return Codes

The SENDTO service return and diagnostic codes indicate the result of the execution. These values are in the ADTRTNCD and ADTDGNCD within the ADT. The diagnostic code is optional and generally indicates a transport provider return code. Read **Appendix C – MESSAGES AND CODES** in the ***CICS Programmer's Toolkit Installation and Administration Guide*** for the return code cross reference table.

This table describes the SEND service return codes:

RETURN	DIAGNOSTIC CODE	DESCRIPTION
CPTIRCOK		Successful.
CPTEVERS		Control block version number is not supported
CPTETOKN		Specified token is not valid
CPTEBUFF		Buffer address and/or length is invalid.
CPTEPRGE	Yes	CPT Interface is terminating
CPTENAPI	Yes	Transport provider API is not available.
CPTETERM	Yes	Environment is being terminated.
CPTERLSE	Yes	Release indication.
CPTEDISC	Yes	Disconnect indication.
CPTEINTG	Yes	Transport provider API integrity error.
CPTEENVR	Yes	Transport provider API environment error.
CPTEFRMT	Yes	Transport provider API format error.
CPTEPROC	Yes	Transport provider API procedure error.
CPTABEND		Abnormal exception occurred.
CPTEOTHR	Yes	An undefined exception occurred.

200801-024040-200100

Example:

In the example, a message is sent to a remote host. The return code is checked to determine SENDTO service completion status.

```
 WORKING-STORAGE SECTION.
 01 CPT-ADT    COPY T09KCADT.
              COPY T09KCRCS.

 LINKAGE SECTION.
 01  MESSAGE  PIC X(32)
          VALUE 'CPT COBOL PROGRAM MESSAGE'.

 PROCEDURE DIVISION.

*
* Identify Service
*
    .
    .
    .
    MOVE 1980 TO ADTRPORT.
    MOVE '127.0.0.1' TO ADTRNAME.

*
* Application and CPT Send Data Transfer processing
*
    SET ADTBUFFA TO ADDRESS OF MESSAGE.
    SET ADTBUFFL TO LENGTH OF MESSAGE.
    CALL 'T09SNTO' USING CPT-ADT.
    IF NOT ADT-RCOKAY
*       Process and log SEND service error
    END-IF.

*   CPT Terminate Endpoint processing
*

*
* Terminate Transaction
*
    EXEC CICS RETURN
    END-EXEC.
```

TAKE

The TAKE service establishes ownership of a connection and associated internal CPT resources. The TAKE service is optional and does not affect an active connection nor does it issue any transport provider requests. This service affects CPT TRUE management routines scheduled on the user's behalf during task termination.

To invoke the TAKE service, a user application is required to first build an AFM and then to issue a call to the TAKE routine. The only valid and required arguments are the AFM version number and the connection token. On completion, a return code indicates success or failure of the request.

This table describes the TAKE service arguments:

COPY NAME	SIZE	CREATED BY
T09KCAFM	34 (X'22')	User application or the LISTEN service

Structure

This is the structure of the TAKE service in COBOL language:

```
   01  CPT-AFM.
                 COPY T09KCAFM.
*****************************************************************
*                              CPT-AFM                          *
*  CICS PROGRAMMER'S TOOLKIT FACILITY MANAGEMENT CONTROL BLOCK*
*---------------------------------------------------------------*
         05  AFMVERS          PIC   9(4) COMP VALUE 2.
             88  AFMVERSN     VALUE 2.
         05  AFMFUNC          PIC   9(4) COMP VALUE 0.
         05  AFMTOKEN         PIC   9(8) COMP VALUE 0.
         05  FILLER           PIC   9(8) COMP VALUE 0.
         05  FILLER           PIC   9(8) COMP VALUE 0.
         05  AFMRTNCD         PIC   9(8) COMP VALUE 0.
             88  AFM-RCOKAY   VALUE  0.
             88  AFM-WARNING  VALUE  1 THRU 15.
             88  AFM-ERROR    VALUE 16 THRU 255.
         05  AFMDGNCD         PIC   9(8) COMP VALUE 0.
         05  FILLER           PIC   9(8) COMP VALUE 0.
         05  FILLER           PIC   9(4) COMP VALUE 0.
```

PARAMETER	DESCRIPTION
AFMVERS	Version Indicates the CPT version number of the argument list used by the calling program. This required field is set to 2 for this release of CPT. Default: 2

200801-024040-200100

PARAMETER	DESCRIPTION
AFMFUNC	Function code Indicates the function or callable service ID requested by the application program. This field is set by the application, but is initialized by the TRUE interface stub program. Default: None
AFMTOKEN	Connection or endpoint token Specifies a token that represents a TCP connection, a TCP listening endpoint, or a UDP endpoint. A token is created by the TCP connection initiation routines or by the UDP data transfer and endpoint creation routines. The token is required. Default: 0
AFMRTNCD	Return code Indicates the return code set by the GIVE service. This value is returned and indicates the success or failure of the service. Default: 0
AFMDGNCD	Diagnostic code Indicates the diagnostic code received by the GIVE service for a transport provider request and is not set by the GIVE service. The GIVE service does not issue transport provider requests; hence, it never sets the diagnostic code. Default: 0

Completion Information

The TAKE service completes normally when all resources associated with this connection are processed.

On normal return to the application program, the general return code in AFMRTNCD is set to zero (CPTIRCOK). The diagnostic code in AFMDGNCD is always zero.

If the TAKE service completes abnormally, then some resources associated with this connection cannot be successfully transferred from one task to another. The general return code in AFMRTNCD and the diagnostic code in AFMDGNCD indicate the nature of the failure. The diagnostic code (AFMDGNCD) is not used by the TAKE service and no information is returned.

Return Codes

The TAKE service return code indicates the result of the execution. This value is in the AFMRTNCD within the AFM. Read **Appendix C – MESSAGES AND CODES** in the ***CICS Programmer's Toolkit Installation and Administration Guide*** for the return code cross reference table.

This table describes the TAKE service return codes:

RETURN	DIAGNOSTIC CODE	DESCRIPTION
CPTIRCOK		Successful.
CPTEVERS		Control block version number is not supported.
CPTETOKN		Specified token is not valid.
CPTENAPI		Transport provider API is not available.
CPTABEND		Abnormal exception occurred.
CPTEOTHR		An undefined exception occurred.

200801-024040-200100

Usage Information

The TAKE service acquires ownership of a connection from one task to another. This service is non-blocking and does not effect any pending transport provider data transfer requests. The association established by the TAKE service lets the CPT properly manage resources during task termination. This ability to give and take ownership of connections offers the user a range of programming options, while still providing CPT with resource management capabilities.

The TAKE service utilizes the AFM. It requires the application to set the AFM version number and token fields. No other AFM fields are referenced.

When a connection is established there are internal CPT resources associated with that connection. CPT is responsible for proper clean-up of those resources on task or transaction termination. These resources include storage allocated by CPT, the API, and the transport provider storage.

A server application is a good example of how the TAKE service benefits a user application. A listening task issues the GIVE service and starts a new transaction to handle data transfer. The data transfer transaction then takes the connection. This sequence prevents a connection from being closed (implicitly by the CPT task termination exit) if the server application terminates. However, if the data transfer transaction is terminated without issuing an explicit close (CPT CLOSE service) an implicit close is scheduled and resource management is handled by the CPT task termination exit.

Additionally, an implicit TAKE facility is implemented with the SEND, RECEIVE, and TRANSLATE services. The task to issue a SEND, RECEIVE, or TRANSLATE service obtains control of the connection and associated resources. The advantage is that the TAKE service is optional. Even though TAKE can be implicit and therefore optional, Interlink recommends that you issue TAKE to avoid having a GIVEn connection not associated with any transactions. Ownership of a connection and resources provide for clean-up processing during abnormal termination.

The version number (AFMVERS) indicates the CPT release level in which this user application program is written. This required field must be set to AFMVERS2 and is validated by the TAKE service before processing the request.

The function code (AFMFUNC) indicates the CPT callable service ID. The field is not initialized by a user application program and has little value to the application except for dump analysis. The function code identifies and maps an argument list with the error or trace log and dump analysis.

The token (AFMTOKEN) indicates the connection and internal resources that are to be processed. This is a required field and is validated by the TAKE service before processing the request.

The AFMOPTCD field specifies TAKE service processing control options, and provides a mechanism for event notification on return to the application program. Currently, this field is not used by application programs.

20-024040-200100

Example:

In this example a data processing transaction retrieves the ACM, issues the TAKE service, and performs some required operation. The token is loaded from the ACM and is used by the TAKE service. The return code is checked to determine TAKE service completion status.

```
WORKING-STORAGE SECTION.
01    CPT-ACM    COPY T09KCACM.
01    CPT-AFM    COPY T09KCAFM.
01    CPT-ADT    COPY T09KCADT.
01    CPT-ACL    COPY T09KCACL.
01    CPT-RCS    COPY T09KCRCS.

PROCEDURE DIVISION.

   EXEC   CICS RETRIEVE FROM(ACM) LENGTH(...)
   END-EXEC.

   MOVE ACMTOKEN TO AFMTOKEN.

   CALL 'T09FTAKE' USING CPT-AFM.
   IF NOT AFM-RCOKAY
*         Process and log TAKE service error
   ELSE
*
*         Application and CPT data transfer (SEND/RECEIVE) processing
*

*
*         CPT Connection Release processing
*
   END-IF.
*
* Terminate Transaction
*
   EXEC CICS RETURN
   END-EXEC.
```

200801-024040-200100

Example:

In this example a data processing transaction retrieves the ACM and performs some required data transfer operation. The token is loaded from the ACM and is used for CPT service requests. The difference between this example and the previous example is that no TAKE service is explicitly issued by the application. The CPT data transfer services, SEND and RECEIVE, implement the TAKE service internally, so an explicit TAKE service request can be omitted.

```
 WORKING-STORAGE SECTION.
 01    CPT-ACM    COPY T09KCACM.
 01    CPT-ADT    COPY T09KCADT.
 01    CPT-ACL    COPY T09KCACL.
 01    CPT-RCS    COPY T09KCRCS.

 PROCEDURE DIVISION.

    EXEC   CICS RETRIEVE FROM(ACM) LENGTH(...)
    END-EXEC.

    MOVE ACMTOKEN TO ADTTOKEN.

*
*   Application and CPT data transfer (SEND/RECEIVE) processing
*

*
*   CPT Connection Release processing
*
* Terminate Transaction
*
    EXEC CICS RETURN
    END-EXEC.
```

020-024040-200100

TRANSLATE

The TRANSLATE service translates data between EBCDIC and ASCII character sets. CPT is customized with a default translation table, but applications can override the default. The TRANSLATE service does not affect an active connection nor issue any transport provider requests.

To invoke the TRANSLATE service, a user application is required to first build an Argument for Data Translation (AXL) and then to issue a call to the TRANSLATE routine. The AXL is required to contain the version number, connection token, user buffer address and length, and type or direction of translation requested. Additional arguments for application specific translation tables are supported. When the TRANSLATE service completes, the buffer contents are converted into the corresponding characters and a return code is generated indicating the status of the request.

This table describes the TRANSLATE service arguments:

COPY NAME	SIZE	CREATED BY
T09KCAXL	32 (X'20')	User application

Structure

This is the structure of the TRANSLATE service in COBOL language:

```
  01  CPT-AXL.
                COPY T09KCAXL.

*****************************************************************
*                          CPT-AXL                              *
*     CICS PROGRAMMER'S TOOLKIT DATA TRANSLATION CONTROL BLOCK*
*---------------------------------------------------------------*
         05  AXLVERS          PIC   9(4) COMP VALUE 2.
             88 AXLVERSN      VALUE 2.
         05  AXLFUNC          PIC   9(4) COMP VALUE 0.
         05  AXLTOKEN         PIC   9(8) COMP VALUE 0.
         05  AXLSADDR         POINTER    VALUE NULL.
         05  AXLSLENG         PIC   9(8) COMP VALUE 0.
         05  AXLRTNCD         PIC   9(8) COMP VALUE 0.
             88 AXL-RCOKAY    VALUE  0.
             88 AXL-WARNING   VALUE  1 THRU  15.
             88 AXL-ERROR     VALUE 16 THRU 255.
         05  AXLDGNCD         PIC   9(8) COMP VALUE 0.

*---------------------------------------------------------------*
*  AXLXMODE         CHARACTER SET MODE                          *
*---------------------------------------------------------------*
     * SINGLE BYTE CHARACTER SET:
             88 AXLXMODE-SBCS VALUE  0.
     * DOUBLE BYTE CHARACTER SET:
             88 AXLXMODE-DBCS VALUE  1.
     * MIXED SBCS/DBCS CHARACTER SET:
             88 AXLXMODE-MIXD VALUE  2.
      * NUMBERS SET:
```

200801-024040-200100

```
            88 AXLXMODE-NUMS VALUE  4.

*-------------------------------------------------------------*
*  AXLXTYPE        TRANSLATION TYPE REQUEST                   *
*-------------------------------------------------------------*
         05  AXLXTYPE         PIC   9(4) COMP VALUE 0.
     * TRANSLATE ASCII-TO-EBCDIC:
            88 AXLXTYPE-ATOE VALUE  1.
     * TRANSLATE EBCDIC-TO-ASCII:
            88 AXLXTYPE-ETOA VALUE  2.
     * TRANSLATE ASCII-TO-UPPER CASE:
            88 AXLXTYPE-AUPC VALUE  4.
     * TRANSLATE EBCDIC-TO-UPPER CASE:
            88 AXLXTYPE-EUPC VALUE  8.

*-------------------------------------------------------------*
*  AXLTABLE        ADDRESS OF USER TRANSLATION TABLE, IF ANY *
*-------------------------------------------------------------*
         05  AXLTABLE         POINTER    VALUE NULL.
```

The following constants are contained in the `T09MAC` library member, `T09KCCON`:

```
01  AXL-CONSTANTS.

*-------------------------------------------------------------*
*  AXLXMODE        CHARACTER SET MODE                         *
*-------------------------------------------------------------*
     * SINGLE BYTE CHARACTER SET:
        05 AXLMODE-SBCS       PIC   9(8) COMP VALUE 0.
     * DOUBLE BYTE CHARACTER SET:
        05 AXLMODE-DBCS       PIC   9(8) COMP VALUE 1.
     * MIXED SBCS/DBCS CHARACTER SET:
        05 AXLMODE-MIXD       PIC   9(8) COMP VALUE 2.
     * NUMBERS SET:
        05 AXLMODE-NUMS       PIC   9(8) COMP VALUE 4.

*-------------------------------------------------------------*
*  AXLXTYPE        TRANSLATION TYPE REQUEST                   *
*-------------------------------------------------------------*
     * TRANSLATE ASCII-TO-EBCDIC:
        05 AXLTYPE-ATOE       PIC   9(8) COMP VALUE 1.
     * TRANSLATE EBCDIC-TO-ASCII:
        05 AXLTYPE-ETOA       PIC   9(8) COMP VALUE 2.
     * TRANSLATE ASCII-TO-UPPER CASE:
        05 AXLTYPE-AUPC       PIC   9(8) COMP VALUE 4.
     * TRANSLATE EBCDIC-TO-UPPER CASE:
        05 AXLTYPE-EUPC       PIC   9(8) COMP VALUE 8.
```

2040-024040-200100

PARAMETER	DESCRIPTION
AXLVERS	Version Indicates the CPT version number of the argument used by the calling program. This required field is set to 2 for this release of CPT. Default: 2
AXLFUNC	Function code Indicates the function or callable service ID requested by the application program. This field is not set by the application, but is initialized by the TRUE interface stub program. Default: None
AXLTOKEN	Connection token Specifies a token that represents a TCP connection or UDP endpoint. Default: 0
AXLSADDR	Source text buffer address Indicates the address of the user data buffer to be translated. This required field is a contiguous segment of storage accessible to the user task. The storage area can be aligned on any boundary convenient to the application program. Default: 0
AXLSLENG	Source text buffer length Indicates the length (in bytes) of user data buffer in the storage area, as identified by the AXLSADDR field. This is a required field. A zero value causes the request to fail. Default: 0
AXLRTNCD	Return code Indicates the return code set by the TRANSLATE service. This value is returned and indicates the success or failure of the service. Default: 0
AXLDGNCD	Diagnostic code Indicates the diagnostic code set by the service request. This value specifies a unique number associated with the return code and identifies the translation error. Default: 0
AXLXMODE	Specifies TRANSLATE service translation mode or character set. ◆ AXLXMODE-SBCS – Indicates single-byte character set translation. ◆ AXLXMODE-DBCS – Indicates double-byte character set translation. This option is currently not supported. ◆ AXLXMODE-MIXD – Indicates mixed mode character set translation. This mode specifies single- and double-byte translation. This option is currently not supported. ◆ AXLXMODE-NUMS – Indicates numeric set translation. This option is currently not supported. Default: AXLXMODE-SBCS

200801-024040-200100

PARAMETER	DESCRIPTION
AXLXTYPE	Specifies TRANSLATE service translation type or direction. ◆ AXLXTYPE-ATOE – Indicates ASCII to EBCDIC translation. ◆ AXLXTYPE-ETOA – Indicates EBCDIC to ASCII translation. ◆ AXLXTYPE-AUPC – Indicates ASCII to uppercase ASCII translation. ◆ AXLXTYPE-EUPC – Indicates EBCDIC to uppercase EBCDIC. translation. Default: None
AXLTABLE	Address of user translation table. Default: None

Completion Information

The TRANSLATE service completes normally when the data is translated into the corresponding character set representation.

On normal return to the application program, the general return code in AXLRTNCD is set to zero (CPTIRCOK). The diagnostic code in AXLDGNCD is set to zero.

If the TRANSLATE service completes abnormally, then an error associated with translation occurred. The general return code in AXLRTNCD and the diagnostic code in AXLDGNCD indicate the nature of the failure.

2-024040-200100

Return Codes

The TRANSLATE service return and diagnostic codes indicate the result of the execution. These values are in the AXLRTNCD and AXLDGNCD within the AXL. Read **Appendix C – MESSAGES AND CODES** in the ***CICS Programmer's Toolkit Installation and Administration Guide*** for the return code cross reference table.

This table describes the TRANSLATE service return codes:

RETURN	DIAGNOSTIC CODE	DESCRIPTION
CPTIRCOK		Successful.
CPTEVERS		Control block version number is not supported.
CPTETOKN		Specified token is not valid.
CPTEBUFF		Buffer address and/or length is invalid.
CPTECHAR		Translation table character set is invalid.
CPTEMODE		Translate mode specification is invalid.
CPTEFRMT		Format or specification error.
CPTENAPI		Transport provider API is not available.
CPTABEND		Abnormal exception occurred.
CPTEOTHR		An undefined exception occurred.

Usage Information

The TRANSLATE service translates data between EBCDIC and ASCII. The requirement for translation is application dependent.

The version number (AXLVERS) indicates the CPT release level in which this user application program is written. This required field must be set to AXLVERS2 and is validated by the TRANSLATE service before it processes the request.

The function code (AXLFUNC) indicates the CPT callable service ID. The field is not initialized by a user application program and has little value to the application except for dump analysis. The function code identifies and maps an argument list with the error or trace log and dump analysis.

The token (AXLTOKEN) indicates the connection associated with this translation request. This field is required. However, no transport provider requests are issued. The token is used for internal logging support requirements. This is a required field and is validated by the TRANSLATE service before processing the request.

The AXLXMODE field specifies the character set mode. This field sets single-, double- or mixed-character set translation. Currently, only single-byte character set translation is supported; it is the default.

200801-024040-200100

The AXLXTYPE field specifies the translation direction. This required field indicates EBCDIC to ASCII, or ASCII to EBCDIC. Additionally, characters can be transacted into the corresponding uppercase values.

Example:

In this example, an established connection sends data to an ASCII host. The data is translated by the application and is sent over the connection. The token, data buffer address and length, and translation type are set in the AXL. The default translation mode of SBCS is selected. The return code is checked to determine TRANSLATE service completion status.

```
 WORKING-STORAGE SECTION.
 01    CPT-ACM    COPY T09KCACM.
 01    CPT-ADT    COPY T09KCADT.
 01    CPT-AXL    COPY T09KCAXL.
 01    CPT-ACL    COPY T09KCACL.
 01    CPT-RCS    COPY T09KCRCS.

 LINKAGE SECTION.
 01    MESSAGE    PIC X(1024)
           VALUE 'WELCOME TO A CICS TEST APPLICATION'.

 PROCEDURE DIVISION.

*
* CPT Connection Management initialization and request
*
    .
    .
    .
    MOVE ACMTOKEN TO AXLTOKEN ADTTOKEN ACLTOKEN.

*
*   CPT EBCDIC to ASCII Data Translation processing
*
    SET AXLXTYPE TO AXLXTYPE-ETOA
    SET AXLBUFFA TO ADDRESS OF MESSAGE.
    MOVE 80 TO AXLBUFFL.
    CALL 'T09FXLAT' USING CPT-AXL.
    IF NOT AXL-RCOKAY
*       Process and log TRANSLATE service error
    END-IF.
*
*   Application and CPT Send Data Transfer processing
*
    SET ADTBUFFA TO ADDRESS OF MESSAGE.
    MOVE 80 TO ADTBUFFL.
    CALL 'T09FSEND' USING CPT-ADT.
    IF NOT ADT-RCOKAY
*       Process and log SEND service error
    END-IF.

*
*   CPT Orderly Connection Release processing
*

* Terminate Transaction
*
    EXEC CICS RETURN
    END-EXEC.
```

Example:

In this example an established connection receives ASCII data. The data is translated from ASCII to EBCDIC before application processing. The token, data buffer address and length, and translation type are set in the AXL. The default translation mode of SBCS is selected. The return code is checked to determine TRANSLATE service completion status.

```
 WORKING-STORAGE SECTION.
 01    CPT-ACM   COPY T09KCACM.
 01    CPT-ADT   COPY T09KCADT.
 01    CPT-ACL   COPY T09KCACL.
 01    CPT-AXL   COPY T09KCAXL.
 01    CPT-RCS   COPY T09KCRCS.

 LINKAGE SECTION.
 01    MESSAGE   PIC X(1024)
          VALUE ' '.

 PROCEDURE DIVISION.

*
* CPT Connection Management initialization and request
*
    .
    .
    .
    MOVE ACMTOKEN TO AXLTOKEN ADTTOKEN ACLTOKEN.

*
*   Application and CPT Send Data Transfer processing
*
    SET ADTBUFFA TO ADDRESS OF MESSAGE.
    MOVE 80 TO ADTBUFFL.
    CALL 'T09FRECV' USING CPT-ADT.
    IF NOT ADT-RCOKAY
*       Process and log RECEIVE service error
    END-IF.

*
*   CPT ASCII to EBCDIC Data Translation processing
*
    SET AXLXTYPE TO AXLXTYPE-ATOE
    SET AXLBUFFA TO ADDRESS OF MESSAGE.
    MOVE 80 TO AXLBUFFL.
    CALL 'T09FXLAT' USING CPT-AXL.
    IF NOT AXL-RCOKAY
*       Process and log TRANSLATE service error
    END-IF.
*
* Terminate Transaction
*
    EXEC CICS RETURN
    END-EXEC.
```

200801-024040-200100

Return Codes

The T09KCRCS copy member is used to resolve API service return codes.

```
*****************************************************************
* EXCEPTION CODES POSTED IN RETURN CODE ARGUMENT FIELDS         *
*---------------------------------------------------------------*
      01  CPT-RETURN-CODES.
          05 FILLER   PIC 9(8) COMP VALUE 40.
      *
          05 CPTIRCOK PIC 9(8) COMP VALUE 0.
           05 FILLER   PIC X(52)
            VALUE 'REQUEST COMPLETED SUCCESSFULLY'.

*****************************************************************
*                     WARNINGS                                  *
*---------------------------------------------------------------*
          05 CPTWTIMO PIC 9(8) COMP VALUE 1.
           05 FILLER   PIC X(52)
            VALUE 'TIMED RECEIVE SERVICE CALL TIMED OUT'.
          05 CPTWNEGO PIC 9(8) COMP VALUE 4.
           05 FILLER   PIC X(52)
            VALUE 'BUFFERS RESET TO SYSTEM LIMITS'.
          05 CPTWBLCK PIC 9(8) COMP VALUE 6.
           05 FILLER   PIC X(52)
            VALUE 'RECEIVE WOULD BLOCK - NO DATA AVAILABLE'.
          05 CPTWNEOM PIC 9(8) COMP VALUE 8.
           05 FILLER   PIC X(52)
             VALUE 'INCOMPLETE DATAGRAM'.
          05 CPTWNSEP PIC 9(8) COMP VALUE 10.
           05 FILLER   PIC X(52)
            VALUE 'NO SEPARATOR CHARACTERS FOUND'.
          05 CPTWEXCP PIC 9(8) COMP VALUE 15.
           05 FILLER   PIC X(52)
            VALUE 'EXCEPTIONAL CONDITION WARNING ISSUED'.

*****************************************************************
*           CONTROL BLOCK ARGUMENT ERRORS                       *
*---------------------------------------------------------------*
          05 CPTEVRSN PIC 9(8) COMP VALUE 17.
           05 FILLER   PIC X(52)
            VALUE 'CONTROL BLOCK VERSION NOT SUPPORTED'.
          05 CPTECONN PIC 9(8) COMP VALUE 18.
           05 FILLER   PIC X(52)
           VALUE 'ERROR IN HOST/PORT/SERVICE SPECIFICATION'.
          05 CPTEPROT PIC 9(8) COMP VALUE 19.
           05 FILLER   PIC X(52)
           VALUE 'SPECIFIED PROTOCOL UNKNOWN/INCOMPATIBLE'.
          05 CPTETOKN PIC 9(8) COMP VALUE 20.
           05 FILLER   PIC X(52)
             VALUE 'SPECIFIED TOKEN IS NOT VALID'.
          05 CPTEBUFF PIC 9(8) COMP VALUE 21.
           05 FILLER   PIC X(52)
           VALUE 'ERROR DETECTED IN BUFFER ADDRESS/LENGTH'.
          05 CPTECHAR PIC 9(8) COMP VALUE 22.
           05 FILLER   PIC X(52)
```

2024040-200100

```
                VALUE 'REQUESTED TRANSLATE CHARACTER SET UNDEFINED'.
               05 CPTEMODE PIC 9(8) COMP VALUE 23.
                05 FILLER   PIC X(52)
                 VALUE 'REQUESTED TRANSLATE MODE UNDEFINED'.
               05 CPTECOPT PIC 9(8) COMP VALUE 24.
                05 FILLER   PIC X(52)
                  VALUE 'REQUESTED CLOSE MODE UNDEFINED'.
               05 CPTETABL PIC 9(8) COMP VALUE 25.
                05 FILLER   PIC X(52)
                VALUE 'REQUESTED TRANSLATE TABLE NON-CONFORMING'.
               05 CPTETRID PIC 9(8) COMP VALUE 26.
                05 FILLER   PIC X(52)
                 VALUE 'REQUESTED TRANSACTION NOT AVAILABLE'.
               05 CPTETIME PIC 9(8) COMP VALUE 27.
                05 FILLER   PIC X(52)
                 VALUE 'RECEIVE TIMEOUT VALUE NOT SPECIFIED'.
               05 CPTESEPC PIC 9(8) COMP VALUE 28.
                05 FILLER   PIC X(52)
               VALUE 'NUMBER OF SEPARATOR CHARACTERS NOT SPECIFIED'.
               05 CPTEOPTN PIC 9(8) COMP VALUE 29.
                05 FILLER   PIC X(52)
                 VALUE 'SPECIFIED RECEIVE OPTIONS IN CONFLICT'.
               05 CPTEOPRL PIC 9(8) COMP VALUE 30.
                05 FILLER   PIC X(52)
                VALUE 'OPTION NOT SUPPORTED BY TRANSPORT PROVIDER'.
               05 CPTEFRMT PIC 9(8) COMP VALUE 31.
                05 FILLER   PIC X(52)
                VALUE 'EXCEPTIONAL SPECIFICATION ERROR DETECTED'.

      ************************************************************
      *               LOCAL ENVIRONMENT ERRORS                   *
      *----------------------------------------------------------*
                05 CPTEPBSY PIC 9(8) COMP VALUE 33.
                 05 FILLER   PIC X(52)
                 VALUE 'REQUESTED LISTEN PORT ALREADY BEING SERVED'.
                05 CPTENAPI PIC 9(8) COMP VALUE 34.
                 05 FILLER   PIC X(52)
                 VALUE 'CPT INTERFACE NOT INITIALIZED - PLEASE RETRY'.
                05 CPTENAVL PIC 9(8) COMP VALUE 35.
                 05 FILLER   PIC X(52)
                  VALUE 'REQUESTED FACILITY UNAVAILABLE'.
                05 CPTEDRAN PIC 9(8) COMP VALUE 36.
                 05 FILLER   PIC X(52)
                 VALUE 'CANNOT HANDLE REQUEST - ENVIRONMENT DRAINING'.
                05 CPTETERM PIC 9(8) COMP VALUE 40.
                 05 FILLER   PIC X(52)
               VALUE 'CANNOT HANDLE REQUEST - ENVIRONMENT TERMINATING'.
                05 CPTEENVR PIC 9(8) COMP VALUE 47.
                 05 FILLER   PIC X(52)
                 VALUE 'INDEFINITE ERROR DETECTED IN THE ENVIRONMENT'.

      ************************************************************
      *                CONNECTION EXCEPTIONS                     *
      *----------------------------------------------------------*
                05 CPTERLSE PIC 9(8) COMP VALUE 65.
                 05 FILLER   PIC X(52)
                  VALUE 'ORDERLY RELEASE OF CONNECTION INITIATED'.
                05 CPTEDISC PIC 9(8) COMP VALUE 68.
                 05 FILLER   PIC X(52)
```

200801-024040-200100

```
           VALUE 'REMOTE CONNECTION TERMINATED OR NOT AVAILABLE'.
          05 CPTEPRGE PIC 9(8) COMP VALUE 72.
           05 FILLER   PIC X(52)
             VALUE 'CONNECTION PURGED'.
          05 CPTEINTG PIC 9(8) COMP VALUE 79.
           05 FILLER   PIC X(52)
            VALUE 'INDEFINITE CONNECTION ERROR DETECTED'.

*****************************************************************
*               OTHER EXCEPTIONS                                *
*---------------------------------------------------------------*
          05 CPTEPROC PIC 9(8) COMP VALUE 143.
           05 FILLER   PIC X(52)
            VALUE 'STATE ERROR - REQUEST CANNOT BE COMPLETED'.
          05 CPTABEND PIC 9(8) COMP VALUE 254.
           05 FILLER   PIC X(52)
             VALUE 'CPT INTERFACE IS NOT AVAILABLE'.
          05 CPTEOTHR PIC 9(8) COMP VALUE 255.
           05 FILLER   PIC X(52)
            VALUE 'INDEFINITE SEVERE ERROR DETECTED'.
      *
     01  CPT-RETCODE-TABLE REDEFINES CPT-RETURN-CODES.

*---------------------------------------------------------------*
*               RETURN CODE TABLE DESCRIPTION                   *
*---------------------------------------------------------------*
         05  CPT-RETCODE-COUNT     PIC  9(8) COMP.
         05  CPT-RETCODE-ARRAY    OCCURS 40 TIMES.
            10  CPT-RETCODE-VALUE  PIC  9(8) COMP.
            10  CPT-RETCODE-DESCR  PIC X(52).
```

CODE NAME	DESCRIPTION
CPTIRCOK	Request completed successfully Indicates the requested service completed successfully.
CPTWTIMO	Timed RECEIVE service call timed out See the following RECEIVE service options: ADTOPTNS-TYPLL ADTOPTNS-TYPSP ADTOPTNS-TMRCV ADTOPTNS-TMPRT
CPTWNEGO	System limits applied to buffer and queue sizes A connection management service returned modified API buffering values. The ACMQSEND, ACMMSEND, ACMQRECV or ACMMRECV values were modified during transport provider connection negotiation. This return code does not generally indicate a serious error, but rather a warning.
CPTWBLCK	RECEIVE would block (no data currently available) No data was available on a TCP connection if the call was to the RECEIVE service, or no datagrams were available at a UDP endpoint if the call was to the RCVFROM service. In both cases, ADTOPTNS was set to ADTOPTNS-NBLKR.

CODE NAME	DESCRIPTION
CPTWNEOM	This is not the whole datagram On a RCVFROM service call, ADTBUFFA for a length of the ADTBUFFL was not large enough to hold the entire datagram. Subsequent RCVFROM calls will be required to retrieve the rest of the datagram.
CPTWNSEP	No separator character found On a RECEIVE service call with the SEP option specified (ADTOPTNS-TYPSP) no indicated separator sequence was found in the data. Check user data and make sure the ADTTIMEO value was long enough to receive all of the data.
CPTWEXCP	Transport provider exceptional condition A transport provider exceptional error has occurred. Review diagnostic code for additional information.
CPTEVERS	Argument version number not supported Indicates the version number specified is not supported. The service request is not executed.
CPTECONN	Requested host/service port connection not found A connection management service request failed due to an invalid or unresolved host, service name or port number specification. The service request is not executed.
CPTEPROT	Specified protocol not supported A connection management service request failed due to an invalid or unsupported protocol specification. The protocol specified was not TCP or UDP. The service request is not executed.
CPTETOKN	Specified token is invalid Specified token in service argument is invalid. The service request is not executed.
CPTEBUFF	Buffer address and/or length invalid Specified buffer address and/or length is invalid. A data transfer or data translation service request failed verification of buffer address or length. The service request is not executed.
CPTECHAR	Translate character set is invalid Specified translation character set is invalid. The translation service request is not executed.
CPTEMODE	Translate mode specification is invalid Specified translation mode is invalid. The Translation service request is not executed.
CPTECOPT	CLOSE mode specification is invalid An invalid or unknown CLOSE option was specified on a call to the CLOSE service.
CPTETABL	Specified translate table not correct An invalid character set table was specified on a call to the TRANSLATE service.
CPTETRID	Designated transaction ID cannot be started The LISTEN service attempted to start a transaction that failed. The LISTEN service request is terminated and the well-known server port is released.

200801-024040-200100

CODE NAME	DESCRIPTION
CPTETIME	RECEIVE service timeout value must be greater than zero Specifying any of these options on a RECEIVE call with an ADTTIMEO=0 will result in CPTETIME being returned in ADTRTNCD: ADTOPTNS-TYPLL ADTOPTNS-TYPSP ADTOPTNS-TMRCV ADTOPTNS-TMPRT
CPTESEP#	RECEIVE type SEP requires # SEP chars = 1 or 2 On a RECEIVE service call with the SEP option specified (ADTTYPSP), ADTSEP# was not equal to 1 or 2.
CPTEOPTN	RECEIVE options combination is invalid On a RECEIVE service call, more than one of these options was specified: ADTOPTNS-NBLKR ADTOPTNS-TYPLL ADTOPTNS-TYPSP ADTOPTNS-TMRCV ADTOPTNS-TMPRT
CPTEOPRL	RECEIVE option not supported by transport carrier You have attempted a CPT function that is not supported by the release of SNS/TCPaccess that you are running.
CPTEFRMT	Other transport provider format or specification error A transport provider format or specification error occurred. Review diagnostic code for additional information. The service request failed to execute successfully.
CPTEPBSY	Selected port is busy with active server The LISTEN service attempted to establish a connection to the local transport provider but the specified port was already allocated. The LISTEN service request failed to execute.
CPTENAPI	API, transport provider or CICS not available The transport provider, API communication subsystem or CICS is not available. Review diagnostic code for additional information. The service request failed to execute.
CPTEDRAN	Transport provider or API is being drained The transport provider or API communication subsystem is being drained. This is an indication that the CPT programming interface may be terminating.
CPTETERM	Transport provider or API is being terminated The transport provider or API communication subsystem is being terminated. This is an indication that the CPT programming interface is terminated. All connections have been closed and no communication is available.
CPTEENVR	Other transport provider environment error A transport provider environment error occurred. Review diagnostic code for additional information. The service request failed to execute successfully.
CPTERLSE	Orderly release of full duplex connection indicated Indicates that the remote host has released its half of the full duplex connection.The connection is still available for communication, but the user application will not receive data for the remote host.

20-024040-200100

CODE NAME	DESCRIPTION
CPTEDISC	Remote connection not available or aborted Indicates that a connection request failed or an established connection was aborted. The connection request is not established or an established connection is terminated.
CPTEPRGE	CICS Programmer's Toolkit interface terminated The CPT interface was terminated. The CPT termination transaction was initiated by either the termination transaction or CICS shutdown PLT entry.
CPTEINTG	Other transport provider integrity error A transport provider integrity error occurred. Review diagnostic code for additional information. The service request failed to execute successfully.
CPTEPROC	Other transport provider procedure error A transport provider procedure error occurred. Review diagnostic code for additional information. The service request failed to execute successfully.
CPTABEND	Abnormal environmental error The service request experienced an abnormal termination condition. The service request failed to execute. The CICS dump data set contains additional information related to the abend.
CPTEOTHR	Any other currently undefined condition An unknown condition was detected. Review the diagnostic code for additional information. The service request failed to execute.

200801-024040-200100

OS/VS COBOL and VS COBOL II Differences

The COBOL copy members and sample programs provided with CPT are written for the COBOL II compiler. If you need to use the old OS/VS COBOL compiler to compile the sample programs, there are several changes that must be made for compatibility with the old compiler.

- Replace `POINTER` definitions in the CPT control blocks.

  The address fields in the ADT and AXL control blocks are defined as `POINTER` fields, so that they can be set with the '`ADDRESS OF`' special register of COBOL II. For the old compiler, these address fields must be changed to a `PIC S9(8) COMP` definition in the `T09KCADT` and `T09KCAXL` copy members.

- Replace `ADDRESS OF` and `LENGTH OF` specifications.

  These special registers can be used with the COBOL II compiler to set the address or length of a storage area. For the old compiler, the `SET` statement must be replaced with a `MOVE` to (or from) a BLL-Cell for an address field, or a `MOVE` to (or from) a halfword or fullword `COMP` field for lengths.

- Use BLL-Cells to manage external addresses.

  One major advantage of the COBOL II compiler was that it eliminated having to manipulate BLL-Cells in CICS programs.

  If you are using the old COBOL compiler for CICS programs, BLL-Cells should be a familiar subject. Read the ***CICS Application Programmer's Reference Manual*** for detailed information. A brief example is given below to show the BLL-Cell definitions that must be added to the Linkage Section, and the `SERVICE RELOAD` statements that must be used after moving a new address to a BLL-Cell.

OS/VS COBOL

```
LINKAGE SECTION.
01 DFHCOMMAREA.
   05 FIELDA            PIC X(25).
   05 BUFFER-PTR        PIC S9(8) COMP.
01 BLL-CELLS.
   05 BLL-CELLS-ADDR    PIC S9(8) COMP.
   05 BLL-BUFFER-ADDR   PIC S9(8) COMP.
01 BUFFER-AREA          PIC X(256).
PROCEDURE DIVISION.
   SERVICE RELOAD BLL-CELLS.
   EXEC CICS GETMAIN
             LENGTH(256)
            SET (BLL-BUFFER-ADDR)
   END-EXEC.
   SERVICE RELOAD BUFFER-AREA.
   MOVE 'MESSAGE' TO BUFFER-AREA.
   MOVE BLL-BUFFER-ADDR TO BUFFER-PTR.
```

VS COBOL II

```
LINKAGE SECTION.
01 DFHCOMMAREA.
   05 FIELDA            PIC X(25).
   05 BUFFER-PTR        USAGE IS POINTER.



01 BUFFER-AREA          PIC X(256).
PROCEDURE DIVISION.

    EXEC CICS GETMAIN
              LENGTH(LENGTH OF BUFFER-AREA)
               SET  (ADDRESS OF BUFFER-AREA)
    END-EXEC.

   MOVE 'MESSAGE' TO BUFFER-AREA.
  SET BUFFER-PTR TO ADDRESS OF BUFFER-AREA.
```

- Replace 88 Level names when used to set field values.

  COBOL II allows the use of 88 Level names to set field values with a statement like: `SET ACLOPT-ABORT TO TRUE`. The old compiler will only allow the 88 names to be used for testing a field for that value. To set the field, you must move one of the constant values defined in `T09KCCON` to the field, such as: `MOVE ACLOPTNS-ABORT TO ACLOPTNS`.

- Replace Structured COBOL statements.

  COBOL II introduced several structured programming features that are used by the sample programs. These statements must be changed to conform to the requirements of the old compiler.

  - Inline `PERFORM` statements, terminated with an `END-PERFORM`.
  - `EVALUATE` statements, terminated with and `END-EVALUATE`.
  - Explicit scope terminators for `IF` statements, the `END-IF`.

200801-024040-200100

CICS Storage Requirements

All CICS storage is shared and allocated from above the 16M line, if possible. Storage requirements for a TCP connection or a UDP endpoint within the CICS address space are:

- `1080 + ((72 + xxxMSEND) * xxxQSEND) + ((72 + xxxMRECV) * xxxQRECV)`

  where `xxx` is either ACM for TCP connections or ADT for UDP endpoints.

Note: For TCP server applications, an additional 1080-byte overhead is required to allocate the listening token/socket. This computation accounts only for the TCP connection that is subsequently established by a listening token/socket or by the storage for a UDP endpoint.

Note: The send buffer queue is allocated only if an application calls the `SEND` or `SENDTO` services. Determining the `xxxMSEND` and `xxxQSEND` values is explained below. The receive buffer queue is allocated only if an application calls the `RECEIVE` or `RCVFRM` services. Determining the `xxxMRECV` and `xxxQRECV` values is explained below.

Note: The receive buffer queue is allocated only if an application calls the `RECEIVE` or `RECVFROM` services. Determining the `xxxMRECV` and `xxxQRECV` values is explained below.

The maximum send and receive data sizes, the maximum send and receive buffer sizes, and the send and receive queues' sizes (maximum number of outstanding requests per endpoint) are determined initially by the `ACPCONFG` macro, `ACFTIB`. Read the ***SNS/TCPaccess Customization Guide***, pages A-70 and A-71, for information. The TCP maximum data size default that can be sent or received is 32K. The maximum data buffer size default is 64K. CPT negotiates with SNS/PCaccess to determine these values:

- Determine final `xxxQSEND` value:

```
If xxxSEND is not specified,
       use the lesser of 4 or the value in DFQSEND
   else
       use the lesser of the value specified in xxxQSEND or
       DFQSEND.
```

- Determine final `xxxQRECV` value:

```
If xxxQRECV is not specified,
       use the lesser of 4 or the value in DFGRECV
   else
       use the lesser of the value specified in xxxQRECV or
       DFQRECV.
```

- Determine final `xxxMSEND` value:

```
If xxxMSEND is not specified,
       use the lesser of 4096 or the value in MXLTSND
   else
```

2024040-200100

```
        use the lesser of the value specified xxxMSEND or
        MXLTSND
    either of which must be less than the following
    computation:
        (the lesser of DFLSEND or 61440) / the final
        xxxQSEND value
    else
        use the quotient from this computation as thefinal
        xxxMSEND value.
```

- Determine final `xxxMRECV` value:

```
If xxxMRECV is not specified,
        use the lesser of 4096 or the value in MXLTRCV
    else
        use the lesser of the value specified in xxxMSEND or
        MXLTRCV
either of which must be less than the following
computation:
(the lesser of DFLRECV or 61440) / the final xxxQRECV value
    else
        use the quotient from this computation as the final
        xxxMRECV value.
```

200801-024040-200100

Chapter 5

PL/1 SUBROUTINE CALLS

This chapter describes the PL/1 subroutine calls of CICS/API. It contains these callable routines:

- CLOSE
- CONNECT
- GIVE
- LISTEN
- RECVFROM
- RECEIVE
- SEND
- SENDTO
- TAKE
- TRANSLATE

It also describes the return codes of the T09KPRCS copy member and the CICS storage requirements for a TCP connection or a UDP endpoint.

Messages are in **Appendix C – MESSAGES AND CODES** in the ***CICS Programmer's Toolkit Installation and Administration Guide***.

200801-024040-200100

CLOSE

The CLOSE service closes an established connection. Both orderly (or graceful) and abortive termination options are supported. The CLOSE service performs all associated functions required for CPT resource clean-up.

To invoke the CLOSE service, a user application is required to first build an Argument for Close (ACL) and then to issue a call to the CLOSE routine. Valid arguments include the ACL version number, connection token, and termination options. On completion, a return code indicates success or failure of the request.

This table describes the CLOSE service arguments:

INCLUDE STATEMENT	SIZE	CREATED BY
T09KPACL	30 (X'1E')	User application

Structure

This is the structure of the CLOSE service in PL/1 language:

```
DCL 1 CPT_ACL,
/*BEGIN %INCLUDE SYSLIB
(T09KPACL)**********************************
/*************************************************************/
/*                          CPT_ACL                          */
/*        CICS PROGRAMMER'S TOOLKIT CLOSE CONNECTION BLOCK */
/*-----------------------------------------------------------*/
5  ACLVERS    FIXED BIN (15) INIT (2),   /* ACL BLOCK VERSION */
5  ACLFUNC    FIXED BIN (15) INIT (0),   /* FUNCTION CODE     */
5  ACLTOKEN   POINTER       INIT(NULL), /* DATA XFER TOKEN   */
5  ACLRSVD1   POINTER       INIT(NULL), /* (RESERVED: IN USE) */
5  ACLRSVD2   FIXED BIN (31) INIT (0),   /* (RESERVED: IN USE) */
5  ACLRTNCD   FIXED BIN (31) INIT (0),   /* RETURN CODE       */
5  ACLDGNCD   FIXED BIN (31) INIT (0),   /* DIAGNOSTIC CODE   */
5  ACLOPRSV     CHAR (3) INIT(LOW(3)),   /* RESERVED OPTIONS  */
5  ACLOPTNS   BIT (8) INIT('00000000'B), /* TERMINATION MODE  */
5  ACLRSVD3   FIXED BIN (15) INIT (0);   /* (RESERVED: IN USE) */
/*END   %INCLUDE SYSLIB
(T09KPACL)**********************************
```

The following constants are contained in the T09MAC library member, T09KPCON:

```
          /*-------------------------------------------*/
          /*        TERMINATION TYPE REQUEST           */
          /*-------------------------------------------*/
ACLOPT_ORDER  BIT (8) INIT('00000000'B), /* ORDERLY RELEASE */
ACLOPT_ABORT  BIT (8) INIT('00000001'B), /* ABORTIVE STOP */
```

200801-024040-200100

PARAMETER	DESCRIPTION
ACLVERS	Version number Indicates the CPT version number of the argument list used by the calling program. This required field is set to a binary 2 for this release of CPT. Default: 2
ACLFUNC	Function code Indicates the function or callable service ID requested by the application program. This field is not set by the application, but is initialized by the TRUE interface stub program. Default: None
ACLTOKEN	Connection or endpoint token Specifies a token that represents a TCP connection, a TCP listening end point, or a UDP end point. A token is created by the TCP connection initiation routines or by the UDP data transfer and endpoint creation routines. **Note:** The token is required. Default: None
ACLRTNCD	Return code Indicates the return code set by the CLOSE service. This value is returned and indicates the success or failure of the service. Default: 0
ACLDGNCD	Diagnostic code Indicates the diagnostic code set by the service request. This value generally indicates a transport provider return code. Default: 0
ACLOPTNS	Specifies CLOSE processing control options. These are the options currently supported: ◆ ACLOPT_ORDER – Indicates a graceful termination. This option implements orderly release of the TCP/IP connection. This is the preferred option for terminating a connection, and is used when processing has completed successfully. ◆ ACLOPT_ABORT – Indicates abortive termination. This option implements a disconnect or reset of the TCP/IP connection. This option is generally used after an unrecoverable application error has occurred. **Note:** The notion of orderly or abortive CLOSE for a UDP endpoint is meaningless and the options specified when calling CLOSE for a UDP token are not important. CPT knows if the token is UDP and will close it properly. Default: ACLOPT_ORDER

20-024040-200100

Completion Information

The CLOSE service completes normally when the connection is terminated and associated resources are released. Graceful termination waits for all pending transport provider asynchronous SEND and RECEIVE requests to complete. Graceful termination also waits for both ends of the full-duplex connection to close. Abortive termination closes the transport provider connection without regard to pending transport provider requests. Abortive termination can cause data loss and should be used only when data integrity is not required.

On normal return to the application program, the general return code in ACLRTNCD is set to zero (CPTIRCOK). The diagnostic code in ACLDGNCD is always zero.

If the CLOSE service completes abnormally, some user data may be lost. The general return code in ACLRTNCD, and the diagnostic code in ACLDGNCD, indicates the nature of the failure. The diagnostic code (ACLDGNCD) may contain a specific code that identifies a particular transport provider error.

Return Codes

The CLOSE service return and diagnostic codes indicate the result of the execution. These values are in the ACLRTNCD and ACLDGNCD within the ACL. The diagnostic code is optional and indicates the transport provider return code. Read **Appendix C – MESSAGES AND CODES** in the ***CICS Programmer's Toolkit Installation and Administration Guide*** for the return code cross reference table.

This table describes the CLOSE service return codes:

RETURN	DIAGNOSTIC CODE	DESCRIPTION
CPTIRCOK		Successful.
CPTEVERS		Control block version number is not supported.
CPTETOKN		Specified token is not valid.
CPTEPRGE	Yes	CPT Interface is terminating.
CPTENAPI	Yes	Transport provider API is not available.
CPTETERM	Yes	Environment is being terminated.
CPTERLSE	Yes	Release indication.
CPTEDISC	Yes	Disconnect indication.
CPTEENVR	Yes	Transport provider API environment error.
CPTEFRMT	Yes	Transport provider API format error.
CPTEPROC	Yes	Transport provider API procedure error.
CPTABEND		Abnormal exception occurred.
CPTEOTHR	Yes	An undefined exception occurred.

200801-024040-200100

Usage Information

The CLOSE service terminates an established transport provider endpoint and releases associated resources. Established transport provider endpoints can be half of a TCP connection, a TCP listening endpoint, or a UDP endpoint, and are represented by a token.

The CLOSE service utilizes the ACL. The CLOSE service requires the application to set the ACL version number and token fields. Optional control information related to termination processing can be specified. The address of the ACL is required to be loaded into register 1 before the CLOSE service.

The version number (ACLVERS) indicates the CPT release level in which this user application program is written. This required field is set to a binary 2 and is validated by the CLOSE service before processing the request.

The function code (ACLFUNC) indicates the CPT callable service ID. The field is not initialized by a user application program and has little value to the application except for dump analysis. The function code identifies and maps an argument list with the error or trace log and dump analysis.

The token (ACLTOKEN) indicates the connection and internal resources that are to be released. This is a required field and is validated by the CLOSE service before processing the request.

The ACLOPTCD field specifies CLOSE processing control options and provides a mechanism for event notification on return to the application program. Currently, ACLOPT_ORDER and ACLOPT_ABORT are the only two options supported; no facility exists for CLOSE event notification, except by way of return code values.

If the option code ACLOPT_ORDER is selected, the CLOSE service completes all pending transport provider requests. These requests represent previous asynchronous SENDs and/or RECEIVEs that have neither completed yet, nor had their completion checked. This may require the CLOSE service to block the application. This option then performs an orderly release of the TCP/IP connection. This is the preferred mechanism for connection termination.

If the option code ACLOPT_ABORT is selected, the CLOSE service terminates the connection and no attempt is made to preserve data in transit. The remote user receives a disconnect indication.

20 024040-200100

Example:

This example establishes a connection, processes data, and closes the connection. The token is loaded from the Argument for Connection Management (ACM) and used by all of the following CPT service requests. The CLOSE service orderly termination option is selected. The return code is checked to determine CLOSE service completion status.

```
SAMP1: PROCEDURE OPTIONS (MAIN);

DCL (T09FCLO)
    ENTRY OPTIONS (INTER ASSEMBLER);
DCL (CPTPRT,MSGPRT) POINTER;

DCL 1  CPT_ACL,
          %INCLUDE SYSLIB(T09KPACL);
DCL 1  CPT_RCS,
          %INCLUDE SYSLIB(T09KPRCS);

%INCLUDE T09KPCON;

   /*
      CPT Connection Management service request
    */

      ADTTOKEN, ACLTOKEN = ACMTOKEN;
   /*
      Application and CPT Data Transfer (SEND/RECV) service request
    */

   /*
      CPT Orderly Connection Release service request
    */

      ACLOPTNS = ACLOPT_ORDER;
      CALL T09FCLO (CPT_ACL);
      IF ACLRTNCD ^= 0
      THEN DO;
      /*
         Process and log CLOSE error
       */
      END;

   /*
      Terminate transaction
    */

      EXEC   CICS RETURN;
```

Example:

This example establishes a connection, processes data, and closes the connection. The token is loaded from the ACM and used by all of the following CPT service requests. The CLOSE service abortive termination option is selected. The return code is checked to determine CLOSE service completion status.

```
SAMP2: PROCEDURE OPTIONS (MAIN);

DCL (T09FCLO)
     ENTRY OPTIONS (INTER ASSEMBLER);
DCL (CPTPRT,MSGPRT) POINTER;

DCL 1  CPT_ACL,
           %INCLUDE SYSLIB(T09KPACL);
DCL 1  CPT_RCS,
           %INCLUDE SYSLIB(T09KPRCS);

%INCLUDE T09KPCON;

    /*
        CPT Connection Management service request
     */

        ADTTOKEN, ACLTOKEN = ACMTOKEN;

    /*
        Application and CPT Data Transfer (SEND/RECV) service request
     */

    /*
        CPT Orderly Connection Release service request
     */

    ORDERLY:

    /*
        CPT Abortive Connection Release service request
     */

    ABORT:
        ACLOPTNS = ACLOPT_ABORT;
        CALL T09FCLO (CPT_ACL);
        IF ACLRTNCD ^= 0
        THEN DO;
        /*
            Process and log CLOSE error
         */
        END;

    /*
        Terminate transaction
     */

        EXEC   CICS RETURN;
```

Example: This example terminates a single-thread server application. A server application contains two CPT connections; the first for the data transfer connection and the second for the server or listening connection. The CLOSE service orderly termination option is selected. The return code is checked to determine CLOSE service completion status.

```
SAMP3: PROCEDURE OPTIONS (MAIN);

DCL (T09FLST,T09FCLO)
    ENTRY OPTIONS (INTER ASSEMBLER);
DCL (CPTPRT,MSGPRT) POINTER;
DCL 1  CPT_ACM,
          %INCLUDE SYSLIB(T09KPACM);
DCL 1  CPT_ACL,
          %INCLUDE SYSLIB(T09KPACL);
DCL 1  CPT_RCS,
          %INCLUDE SYSLIB(T09KPRCS);
%INCLUDE T09KPCON;
   /*
      CPT LISTEN Connection Management service request
    */
      ADTTOKEN, ACLTOKEN = ACMTOKEN;
   /*
      Application and CPT Data Transfer (SEND/RECV) service request
    */
   /*
      CPT Data Transfer Token Orderly Connection Release service request
    */
      ACLOPTNS = ACLOPT_ORDER;
      CALL T09FCLO (CPT_ACL);
      IF ACLRTNCD ^= 0
      THEN DO;
      /*
         Process and log CLOSE error
       */
      END;
   /*
      CPT LISTEN Token Orderly Connection Release service request
    */
      ACLTOKEN = ACMTLSTN;
      ACLOPTNS = ACLOPT_ORDER;
      CALL T09FCLO (CPT_ACL);
      IF ACLRTNCD ^= 0
      THEN DO;
      /*
         Process and log CLOSE error
       */
      END;
   /*
      Terminate transaction
    */

      EXEC   CICS RETURN;
```

200801-024040-200100

CONNECT

The CONNECT service provides a client facility for use by an application program. The CONNECT service establishes a session with the local transport provider, and then actively connects to a server. When a connection is established with a server, the CONNECT service returns control to the calling program. Information related to the connection is updated and returned within the ACM.

To invoke the CONNECT service, a user application is required to first build an ACM and then to issue a call to the CONNECT routine. The minimum information required by this service is version number, server host address and well-known port. Optional information related to data transfer buffering, CPT statistics and tracing, and subtask initialization can be specified.

This table describes the CLOSE service arguments:

INCLUDE STATEMENT	SIZE	CREATED BY
T09KPACM	676 (X'2A4')	User application

Structure

This is the structure of the CONNECT service in PL/1 language:

```
DCL 1 CPT_ACM,
/*BEGIN %INCLUDE SYSLIB
(T09KPACM)**********************************
/*************************************************************/
/*                              CPT_ACM                      */
/*     CICS PROGRAMMER'S TOOLKIT CONNECTION MANAGEMENT BLOCK
/*-------------------------------------------------------------*/
5  ACMVERS    FIXED BIN (15) INIT(2),    /* ACM BLOCK VERSION  */
5  ACMFUNC    FIXED BIN (15) INIT(0),    /* FUNCTION CODE      */
5  ACMTOKEN   POINTER  INIT(NULL),       /* DATA XFER TOKEN    */
5  ACMRSVD1   POINTER  INIT(NULL),       /* (RESERVED: IN USE) */
5  ACMRSVD2   FIXED BIN (31) INIT(0),    /* (RESERVED: IN USE) */
5  ACMRTNCD   FIXED BIN (31) INIT(0),    /* RETURN CODE        */
5  ACMDGNCD   FIXED BIN (31) INIT(0),    /* DIAGNOSTIC CODE    */
5  ACMSFILL   CHAR  (3) INIT(LOW(3)),    /* RESERVED STATS     */
5  ACMSTATS   BIT (8) INIT('00000000'B), /* STATISTICS REQS    */
5  ACMTFILL   CHAR  (2) INIT(LOW(2)),    /* RESERVED TRACE     */
5  ACMTRAC2   BIT (8) INIT('00000000'B), /* TRACE OPTIONS 2    */
5  ACMTRAC1   BIT (8) INIT('00000000'B), /* TRACE OPTIONS 1    */
5  ACMQSEND   FIXED BIN (31) INIT(0),    /* SEND QUEUE SIZE    */
5  ACMMSEND   FIXED BIN (31) INIT(0),    /* SEND MAX BUFFER    */
5  ACMQRECV   FIXED BIN (31) INIT(0),    /* RECV QUEUE SIZE    */
5  ACMMRECV   FIXED BIN (31) INIT(0),    /* RECV MAX BUFFER    */
5  ACMTLSTN   FIXED BIN (31) INIT(0),    /* LISTEN TOKEN       */
5  ACMUCNTX   FIXED BIN (31) INIT(0),    /* USER CONTEXT FIELD */
5  ACMTRNID   CHAR   (4) INIT('  '),     /* SERVER TRANSID FOR */
                                         /* LISTEN TO START    */
5  ACMRSVD4   CHAR   (1) INIT(LOW(1)),   /* (RESERVED: IN USE) */
5  ACMRSVD5   CHAR   (3) INIT(LOW(3)),   /*                    */
5  ACMLPORT   FIXED BIN (15) INIT(0),    /* LOCAL PORT NO.     */
```

```
5  ACMRPORT   FIXED BIN (15) INIT(0),    /* REMOTE PORT NO.     */
5  ACMSRVCE   CHAR  (36) INIT('  '),     /* SERVICE NAME        */
5  ACMRSVD6   CHAR   (1) INIT(LOW(1)),   /* (RESERVED: IN USE)  */
5  ACMRSVD7   CHAR   (1) INIT(LOW(1)),   /*                     */
5  ACMOPTNS   FIXED BIN (15) INIT(0),    /* OPTION CODES        */
5  ACMLADDR   FIXED BIN (31) INIT(0),    /* LOCAL HOST ADDRESS  */
5  ACMRADDR   FIXED BIN (31) INIT(0),    /* REMOTE HOST ADDR    */
5  ACMLNAME   CHAR (255) INIT('  '),     /* LOCAL HOST NAME     */
5  ACMRSVD8   CHAR   (1) INIT(LOW(1)),   /* (RESERVED: IN USE)  */
5  ACMRNAME   CHAR (255) INIT('  '),     /* REMOTE HOST NAME    */
5  ACMRSVD9   CHAR   (1) INIT(LOW(1)),   /* (RESERVED: IN USE)  */
5  ACMRSVDA   FIXED BIN (15) INIT(0),    /* (RESERVED: IN USE)  */
5  ACMRSVDB   FIXED BIN (15) INIT(0),    /* (RESERVED: IN USE)  */
5  ACMTIMEO   FIXED BIN (31) INIT(0),    /* TIMEOUT VALUE       */
5  ACMRSVDC   FIXED BIN (31) INIT(0);    /* RESERVED            */

/*END   %INCLUDE SYSLIB
(T09KPACM)**********************************
```

The following constants are contained in the `T09MAC` library member, `T09KPCON`:

```
        /*-------------------------------------------*/
          /*        STATISTICS LOG REQUESTS          */
        /*-------------------------------------------*/
ACMSTATS_CONN BIT (8) INIT('00000001'B), /*   CONNECTIONS   */
ACMSTATS_TERM BIT (8) INIT('00000010'B), /* TRMNATION STATS */

        /*-------------------------------------------*/
          /*           TRACE LOG REQUESTS            */
        /*-------------------------------------------*/
ACMTRAC1_NTRY BIT (8) INIT('00000001'B), /*   ENTRY POINTS  */
ACMTRAC1_ARGS BIT (8) INIT('00000010'B), /*    ARGUMENTS    */
ACMTRAC1_RECV BIT (8) INIT('00000100'B), /*      TRECV      */
ACMTRAC1_SEND BIT (8) INIT('00001000'B), /*      TSEND      */
ACMTRAC1_TERM BIT (8) INIT('00010000'B), /*   TERMINATION   */
ACMTRAC1_PASS BIT (8) INIT('00100000'B), /*    GIVE/TAKE    */
ACMTRAC1_CLSE BIT (8) INIT('01000000'B), /*     CLOSE       */
ACMTRAC1_TERR BIT (8) INIT('10000000'B), /*    TPLERRORS    */
ACMTRAC2_TOKN BIT (8) INIT('00000001'B), /*     TOKENS      */
ACMTRAC2_TPL  BIT (8) INIT('00000010'B), /*       TPL       */
ACMTRAC2_RLSE BIT (8) INIT('00000100'B), /*     RELEASE     */
ACMTRAC2_STOR BIT (8) INIT('00001000'B), /*     STORAGE     */
ACMTRAC2_CLTD BIT (8) INIT('00010000'B), /*   CLIENT-DATA   */

        /*-------------------------------------------*/
          /*             ACM OPTIONS                 */
        /*-------------------------------------------*/
ACMOPTN1_SYNC  BIT (8) INIT('00000001'B), /* LISTEN SYNCPOINT*/
ACMOPTN1_LTRAN BIT (8) INIT('00000010'B), /* DYNAM TRANSACTN */
ACMOPTN1_NODNR BIT (8) INIT('00000100'B), /* NO DNR NAMES    */
```

200801-024040-200100

PARAMETER	DESCRIPTION
ACMVERS	Version Indicates the version number of the CPT argument list used by the calling program. This required field is set to a binary 2 for this release of CPT. Default: None
ACMFUNC	Function code Indicates the function or callable service ID requested by the application program. This field is not set by the application, but is initialized by the TRUE *i*nterface stub program. Default: None (generated by service stub)
ACMTOKEN	Data transfer token Specifies the token is created and returned by the CONNECT service. This token is used for all subsequent service calls for the client connection. Default: 0 (token returned)
ACMRTNCD	Return code Indicates the return code set by the CONNECT service. This value is returned and indicates the success or failure of the service. Default: 0
ACMDGNCD	Diagnostic code Indicates the diagnostic code received by the CONNECT service for a transport provider request. A detailed explanation of this value can be found in the transport provider's ***API Programmer's Reference Guide***. Default: 0
ACMSTATS	ACMSTATS_CONN | ACMSTATS_TERM Specifies statistics logging options for the application program. This facility can be used for debugging and tuning during development. ◆ ACMSTATS_CONN – Specifies that a message(s) be generated when either a listen service or a data transfer connection is established. These messages are generated by the CPT LISTEN and CONNECT services. The message numbers associated with this option are CPT802I and CPT803I. ◆ ACMSTATS_TERM – Specifies that a message(s) be generated on terminating an established connection. These messages are generated by the CPT CLOSE service. The message numbers associated with this option are CPT807I, CPT808I, and CPT809I. Default: 0 (no statistics logging)

20-024040-200100

PARAMETER	DESCRIPTION
ACMTRAC1	Specifies trace logging options for the application program. This facility can be used for debugging during development. ◆ ACMTRAC1_NTRY – Specifies that a message be generated on entry to a CPT service routine. Messages are generated by all CPT service routines. The message number associated with this option is CPT901I. ◆ ACMTRAC1_ARGS – Specifies that a hex dump of the caller's arguments be generated on entry and exit of a CPT service routine. Messages are generated by the corresponding service routine. The message numbers associated with this option are CPT902I, CPT903I, CPT904I, CPT911I, CPT912, CPT921I, CPT922I, CPT924I, CPT925I, CPT930I, and CPT932I. ◆ ACMTRAC1_RECV – Specifies that a hex dump of the transport provider's input (RECEIVE) data be logged. Messages are generated by the CPT RECV service. The message number associated with this option is CPT913I. ◆ ACMTRAC1_SEND – Specifies that a hex dump of the transport provider's output (SEND) data be logged. Messages are generated by the CPT SEND service. The message number associated with this option is CPT914I. ◆ ACMTRAC1_TERM – Specifies that a message be generated on termination of a CPT service routine. Messages are generated by all CPT service routines. The message number associated with this option is CPT908I. ◆ ACMTRAC1_PASS – Specifies that a hex dump of resources related to a passed connection be logged. Messages are generated by the CPT GIVE and TAKE services. The message numbers associated with this option are CPT919I and CPT920I. ◆ ACMTRAC1_CLSE – Specifies that a hex dump of resources related to a CLOSE processing of a connection be logged. Messages are generated by the CPT CLOSE service. The message number associated with this option is CPT906I ◆ ACMTRAC1_TERR – Specifies that a hex dump of a transport provider API parameter list that fails successful completion be logged. The message number associated with this option is CPT400I.

200801-024040-200100

PARAMETER	DESCRIPTION
ACMTRAC2	◆ ACMTRAC2_TOKN – Specifies that a hex dump of the connection token be logged. Messages are generated by all services routines on entry. The message number associated with this option is CPT909I. ◆ ACMTRAC2_TPL – Specifies that a hex dump of the transport provider API parameter list be logged. The message numbers associated with this option are CPT917I and CTP935I. ◆ ACMTRAC2_RLSE – Specifies that a message indicating a transport provider release indication is to be logged. Messages are generated by various CPT service routines. The message number associated with this option is CPT414I. ◆ ACMTRAC2_STOR – Specifies that a hex dump of storage management argument to be logged. Messages are generated by various CPT service routines. The message numbers associated with this option are CPT928I and CPT929I. ◆ ACMTRACE2_CLTD – Trace transient data writes from the LISTEN service (used with the ACMOPTNS_LTRAN client-data option). The message number associated with this option is CPT918I. Default: 0 (no trace logging)
ACMQSEND	API send queue size Specifies the maximum number of uncompleted SEND requests that can be queued by the application to the transport provider (API). This value lets applications control output processing and can affect throughput rates. The value is negotiated with and can be modified by the transport provider. The total allocation for output processing is the product of the SEND queue and buffer size values and cannot exceed 61K. Default: 4
ACMMSEND	API send buffer size Specifies the maximum number of user data bytes that can be transferred by the application in a single SEND request to the transport provider (API). This value lets applications control output processing and can affect throughput rates. The value is negotiated with and can be modified by the transport provider. The total allocation for output processing is the product of the SEND queue and buffer size values and cannot exceed 61K. Default: 4096
ACMQRECV	API receive queue size Specifies the maximum number of uncompleted RECEIVE requests that can be queued by the application to the transport provider (API). This value lets applications control input processing and can affect throughput rates. The value is negotiated with and can be modified by the transport provider. The total allocation for input processing is the product of the RECEIVE queue and buffer size values and cannot exceed 61K. Default: 4
ACMMRECV	API receive buffer size Specifies the maximum number of user data bytes that can be transferred by the application, in a single RECEIVE request, to the transport provider (API). This value lets applications control input processing and can affect throughput rates. The value is negotiated with and can be modified by the transport provider. The total allocation for input processing is the product of the RECEIVE queue and buffer size values and cannot exceed 61K. Default: 1024

20-024040-200100

PARAMETER	DESCRIPTION
ACMTLSTN	Listen service token This field is not used by the CONNECT service. The value in this field is not validated nor is it modified. Default: None
ACMUCNTX	One word of user context Specifies one arbitrary word of user context to be associated with the connection. The information provided is not interpreted by CPT, and is merely saved with other connection information. Default: 0 (no user context)
ACMTRNID	Listen start transaction ID This field is not used by the CONNECT service. The value in this field is not validated nor is it modified. Default: None
ACMLPORT	Listen well-known service port This value represents the TCP port on the local host that was assigned to the client application by TCP. It is returned to the caller of the CONNECT service. This field is an unsigned positive integer with a maximum value of 65,534. Default: None
ACMRPORT	Remote well-known service port Indicates the remote transport layer address or port. This value represents the TCP port on the remote host to which the client application is trying to connect. It must be filled in by the calling client application unless the ACMSRVCE field is specified. If ACMSRVCE is specified, ACMRPORT will be filled in with the resolved remote port number before returning to the caller of the CONNECT service. This field is an unsigned positive integer with a maximum value of 65,534. Default: None
ACMSRVCE	Transport layer service name Indicates the local transport layer service name. This string is mapped into a transport layer address or port, through internal Domain Name Resolution (DNR). The resolved value will be the well-known port to which the CONNECT service attempts to establish a connection. This field has a maximum length of 36 bytes. This field is optional and is not modified by the CONNECT service. Default: None
ACMOPTNS	TCP connection initialization options. ◆ ACMOPTN1_NODNR – DNR Suppression option. Skip internal DNR calls to resolve and return the remote IP address into an IP name in the ACMRNAME field. If an application is designed such that TCP connection establishment and release happens frequently, this option can save processing time. ◆ ACMOPTN1_LTRAN – Client-Data Listener option. This option is for the LISTEN service and is not validated or modified by the CONNECT service. ◆ ACMOPTN1_SYNC – Listen Syncport option. This option is for the LISTEN service and is not validated or modified by the CONNECT service. Default: None

200801-024040-200100

PARAMETER	DESCRIPTION
ACMLADDR	Local IP host address Indicates the local host internet address. This field is an unsigned four-byte integer value. The local host internet address is updated when a server connection is established, and is returned to the caller. Default: None
ACMRADDR	Remote IP host address Indicates the remote host internet address. Either this field or the remote host name (ACMRNAME) field must be specified. This field is an unsigned four-byte integer value. The remote host internet address is updated when a server connection is established, and is returned to the caller. Default: None
ACMLNAME	Local IP host name Indicates the local host internet name. This field is a 255-byte character string that is padded with blanks. The local host internet name is updated when a client connection is established, and is returned to the caller. Default: None
ACMRNAME	Remote IP host name Indicates the remote host internet name. Either this value or the remote IP address (ACMRADDR) field must be specified. This field is a 255-byte character string that is padded with blanks. The remote host internet name is updated when a server connection is established, and is returned to the caller. Default: None
ACMTIMEO	Client-Data Listener timeout value. This field is optionally used by the LISTEN service and is not validated or modified by the CONNECT service. Default: 1 second

2024040-200100

NETWORK Considerations

The ACM is a common data structure used for both client and server connection initialization. There are common and unique values specified for a particular service request.

This table describes network considerations for PL/1 API:

NAME	SERVER CONDITIONS FOR LISTEN	CLIENT CONDITIONS FOR CONNECT
ACMTLSTN	Listen token returned to user application.	
ACMTRNID	Listen START transaction ID.	
ACMLPORT	Local server or listening transport provider well-known port selected by user application.	Local assigned transport provider port returned to user application.
ACMRPORT	Remote client transport provider port returned to user application.	Remote server transport provider well-known port selected by user application.
ACMSRVCE	Local server transport provider service name selected by user application.	Remote server transport provider service name selected by user application.
ACMRADDR	Remote IP host address returned to user application.	Remote IP host address selected or returned to user application.
ACMLNAME	Local IP host name returned to user application.	Remote IP host name returned to user application.
ACMRNAME	Remote IP host name returned to user application.	Remote IP host name selected or returned to user application.
ACMTIMEO	Client-Data Listener timeout value.	

Completion Information

The CONNECT service completes normally when a connection with a server is established. The CONNECT service initializes the client environment with the transport provider (API) and actively contacts a server and updates connection information within the ACM. Establishing a client connection is represented by storage and is referred to as the token. When a connection is successfully established the ACM to be updated with information related to the connection.

The ACM is updated with information related to the new connection. The local and remote port, IP address, and host names are resolved. Negotiated transport provider SEND and RECEIVE buffering values are returned. The ACM return code ACMRTNCD must be checked to determine the success or failure of the CONNECT service. A zero (0) return code indicates a successful connection.

The return and diagnostic codes should be interpreted by the application to determine the reason for failure. Errors indicating CPT, the transport provider (API), or CICS termination are minor. Interrogate errors for level of severity.

Return Codes

The CONNECT service return and diagnostic codes indicate the result of the execution. These values are in the ACMRTNCD and ACMDGNCD within the ACM. The diagnostic code generally indicates the transport provider return code. Read **Appendix C – MESSAGES AND CODES** in the ***CICS Programmer's Toolkit Installation and Administration Guide*** for the return code cross reference table.

This table describes the CONNECT service return codes:

RETURN	DIAGNOSTIC CODE	DESCRIPTION
CPTIRCOK		Successful.
CPTEVERS		Control block version number is not supported.
CPTECONN	Yes	Requested host/service/port is not found.
CPTEPROT		Specified protocol is not supported.
CPTENAPI	Yes	Transport provider API is not available.
CPTETERM	Yes	Environment is being terminated.
CPTERLSE	Yes	Release indication.
CPTEDISC	Yes	Disconnect indication.
CPTEPRGE	Yes	CPT Interface terminating.
CPTEINTG	Yes	Transport provider API integrity error.
CPTEENVR	Yes	Transport provider API environment error.
CPTEFRMT	Yes	Transport provider API format error.
CPTEPROC	Yes	Transport provider API procedure error.
CPTABEND		Abnormal exception occurred.
CPTEOTHR	Yes	An undefined exception occurred.

20-024040-200100

Usage Information

The CONNECT service lets user-written application programs implement TCP/IP client facilities. The CONNECT service generalized parameter list (ACM) describes the application's communications requirements as well as information related to established connections. On completion, the ACM contains fields initialized by both a user application and by the CONNECT service.

There are required and optional fields initialized by a user or calling application. The ACM version number is required. The server must be identified by the calling program. The server is specified by selecting the remote IP address (ACMRADDR) or host name (ACMRNAME) fields, and the remote port (ACMRPORT) or service name (ACMSRVCE). The selection of port or service name defines the server well-known port address. Optional fields control data transfer buffering, statistics, tracing, and subtask initialization.

On completion of the CONNECT service, the ACM contains information related to the established connection. A token which identifies the connection is returned in the ACM, and must be used in all subsequent requests that refer to the connection. The user application program should make no assumptions regarding the format of a token, other than that it is an unsigned, full word value. Information related to the negotiated buffer values, host names, host addresses, and transport provider addresses are returned in the ACM.

The version number (ACMVERS) indicates the CPT release level in which this user application program is written. This required field is set to a binary 2 and is validated by the CONNECT service before processing the request.

The function code (ACMFUNC) indicates the CPT callable service ID. The field is initialized by the CPT service stub program and has little value to the application except for dump analysis. The function code can identify and map an argument with the error or trace logs, and dump analysis.

The remote IP address (ACMRADDR) or remote host name (ACMRNAME) is required. These fields identify the host to which the CONNECT service initiates a connection request. The IP address has precedence over host name. This implies that the host name field is only used if a IP address is not specified.

The transport provider port number (ACMRPORT) or service name (ACMSRVCE) is required. These fields identify the well-known port to which the CONNECT service initiates a connection request. The port number has precedence over service name. This implies that the service name field is only used if a port number is not specified.

User application programs can control CPT and transport provider data transfer buffering. The ACMQSEND, ACMMSEND, ACMQRECV, and ACMMRECV specify the number and size of buffers allocated. The SEND and RECEIVE buffers are allocated on initial entry into either the SEND or RECEIVE service. The corresponding values used by the SEND and RECEIVE services are independent of each other. The SEND service multiplies the queue and buffer values to determine output storage requirements. The RECEIVE service performs a similar function to determine input storage requirements. The product of the queue and buffer values cannot exceed 61,440. CPT requires some additional storage to manage these buffers. This extra storage is included in the allocation.

200801-024040-200100

The SEND service uses the ACMQSEND value: the RECEIVE service uses the ACMQRECV value. These values indicate the maximum number of uncompleted SEND or RECEIVE requests which can be queued by the application to the transport provider. The CPT data transfer services schedule asynchronous transport provider or API requests on the caller's behalf. These API requests must be completed before they can be used again. If the queue values are too small, the CPT data transfer service may block the caller's request and schedule a WAIT command within the service routines. If the queue values are too large, the user application may be wasting storage.

The SEND service uses the ACMMSEND value; the RECEIVE service uses the ACMMRECV value. These values indicate the maximum number of user data bytes that can be transferred by the application in a single SEND or RECEIVE request to the transport provider. The user application is not limited to these values within the data transfer services. However, it is important to note that multiple transport provider or API requests are issued to complete the caller's request. Information on queue and buffer size can be found in the SEND and RECEIVE service description section in this chapter.

Initially, the tuning of data transfer storage may not be a concern; however, the ability to control storage allocation can prove beneficial to the application or CICS region. Additionally, queue size can increase data transfer throughput. Consider enabling the statistics option to gather CPT statistical information, which can be used to set the SEND or RECEIVE queue and buffer size values.

The CONNECT service can modify data transfer buffer allocation values. These values are negotiated with the transport provider and, depending on the site configuration, can be reduced. Any application dependent on these values should check them on return. These values are generally not modified when giving reasonable numbers. However, it is advisable check with the site administrator for the maximum values of the API transport services.

A number of arguments are not set by the calling application, but are returned to the caller. These values represent information related to the client connection and can be used by the application. The local port, host name, and IP address are returned as well as the client's corresponding values.

20-024040-200100

Example:

This example establishes a client connection, processes data, and closes the connection. The connection is established to LOCALHOST and the server well-known port is 1234. The token is loaded from the ACM and used by all of the following CPT service requests. The return code is checked to determine CONNECT service completion status.

```
SAMP1: PROCEDURE OPTIONS (MAIN);

DCL (T09FCON)
    ENTRY OPTIONS (INTER ASSEMBLER);
DCL (CPTPRT,MSGPRT) POINTER;

DCL 1  CPT_ACM,
          %INCLUDE SYSLIB(T09KPACM);
DCL 1  CPT_RCS,
       %INCLUDE SYSLIB(T09KPRCS);

%INCLUDE T09KPCON;

   /*
      CPT CONNECT Connection Management service request
    */
      ACMRPORT = 1234;
      ACMRNAME = 'LOCALHOST'';d

      CALL T09FCON (CPT_ACM);
      IF ACMRTNCD ^= 0
      THEN DO;
      /*
          Process and log CONNECT error
          Terminate transaction
       */
      END;

      ADTTOKEN, ACLTOKEN = ACMTOKEN;

   /*
      Application and CPT Data Transfer (SEND/RECV) service request
    */

   /*
      CPT Connection Release service request
    */

   /*
      Terminate transaction
    */

      EXEC   CICS RETURN;
```

200801-024040-200100

Example:

This example establishes a client connection, processes data, and closes the connection. The connection is established to LOCALHOST and the port is mapped into service name ECHO. The token is loaded from the ACM and used by all of the following CPT service requests. The return code is checked to determine CONNECT service completion status.

```
SAMP2: PROCEDURE OPTIONS (MAIN);

DCL (T09FCON)
     ENTRY OPTIONS (INTER ASSEMBLER);
DCL (CPTPRT,MSGPRT) POINTER;

DCL 1  CPT_ACM,
          %INCLUDE SYSLIB(T09KPACM);
DCL 1  CPT_RCS,
          %INCLUDE SYSLIB(T09KPRCS);
%INCLUDE T09KPCON;
    /*
       CPT CONNECT Connection Management service request
     */
       ACMSRVCE = 'ECHO';
       ACMRNAME = 'LOCALHOST'';
       CALL T09FCON (CPT_ACM);
       IF ACMRTNCD ^= 0
       THEN DO;
       /*
          Process and log CONNECT error
          Terminate transaction
        */
       END;
       ADTTOKEN, ACLTOKEN = ACMTOKEN;
    /*
       Application and CPT Data Transfer (SEND/RECV) service request
     */
    /*
       CPT Connection Release service request
     */
    /*
       Terminate transaction
     */
       EXEC   CICS RETURN;
```

GIVE

The GIVE service releases ownership of a connection and associated internal CPT resources. The GIVE service is optional and does not affect an active connection, nor does it issue any transport provider requests. This service affects CPT TRUE management routines, scheduled on the user's behalf during task termination.

To invoke the GIVE service, a user application is required to first build an Argument for Facility Management (AFM) and then to issue a call to the GIVE routine. The only valid and required arguments are the AFM version number and the connection token. On completion, a return code is set to indicate the success or failure of the request.

This table describes the GIVE service arguments:

INCLUDE STATEMENT	SIZE	CREATED BY
T09KPAFM	34 (X'22')	User application or the LISTEN service.

Structure

This is the structure of the GIVE service in PL/1 language:

```
DCL 1 CPT_AFM,
/*BEGIN %INCLUDE SYSLIB
(T09KPAFM)**********************************
/*************************************************************/
/*                          CPT_AFM                          */
/*   CICS PROGRAMMER'S TOOLKIT FACILITY MANAGEMENT CONTROL BLOCK
/*-----------------------------------------------------------*/
5  AFMVERS    FIXED BIN (15) INIT(2),    /* AFM BLOCK VERSION  */
5  AFMFUNC    FIXED BIN (15) INIT(0),    /* FUNCTION CODE      */
5  AFMTOKEN   POINTER  INIT(NULL),       /* DATA XFER TOKEN    */
5  AFMRSVD1   POINTER  INIT(NULL),       /* RESERVED           */
5  AFMRSVD2   FIXED BIN (31) INIT(0),    /* RESERVED           */
5  AFMRTNCD   FIXED BIN (31) INIT(0),    /* RETURN CODE        */
5  AFMDGNCD   FIXED BIN (31) INIT(0),    /* DIAGNOSTIC CODE    */
5  AFMRSVD3   FIXED BIN (31) INIT(0),    /* RESERVED           */
5  AFMRSVD4   FIXED BIN (15) INIT(0);    /* RESERVED           */
```

PARAMETER	DESCRIPTION
AFMVERS	Version Indicates the CPT version number of the argument list used by the calling program. This required field is set to a binary 2 for this release of CPT. Default: None

200801-024040-200100

PARAMETER	DESCRIPTION
AFMFUNC	Function code Indicates the function or callable service ID requested by the application program. This field is not set by the application, but is initialized by the TRUE interface stub program. Default: None
AFMTOKEN	Connection or endpoint token Specifies a token that represents a TCP connection, a TCP listening endpoint, or a UDP endpoint. A token is created by the TCP connection initiation routines or by the UDP data transfer and endpoint creation routines. The token is required. Default: 0
AFMRTNCD	Return code Indicates the return code set by the GIVE service. This value is returned and indicates the success or failure of the service. Default: 0
AFMDGNCD	Diagnostic code Indicates the diagnostic code received by the GIVE service for a transport provider request and is not set by the GIVE service. The GIVE service does not issue transport provider requests; hence, it never sets the diagnostic code. Default: 0

Completion Information

The GIVE service completes normally when all resources associated with this connection are processed.

On normal return to the application program, the general return code in AFMRTNCD is set to zero (CPTIRCOK). The diagnostic code in AFMDGNCD is always zero.

If the GIVE service completes abnormally, some resources associated with this connection cannot be successfully transferred from one task to another. The general return code in AFMRTNCD and the diagnostic code in AFMDGNCD indicate the nature of the failure. The diagnostic code (AFMDGNCD) is not used by the GIVE service and no information is returned.

20-024040-200100

Return Codes

The GIVE service return and diagnostic codes indicate the result of the execution. These values are in the AFMRTNCD and AFMDGNCD within the AFM. Read **Appendix C – MESSAGES AND CODES** in the ***CICS Programmer's Toolkit Installation and Administration Guide*** for the return code cross reference table.

This table describes the GIVE service return codes:

RETURN	DESCRIPTION
CPTIRCOK	Successful.
CPTEVERS	Control block version number is not supported.
CPTETOKN	Specified token is not valid.
CPTENAPI	Transport provider API is not available.
CPTABEND	Abnormal exception occurred.
CPTEOTHR	An undefined exception occurred.

200801-024040-200100

Usage Information

The GIVE service releases ownership of a connection. This service is non-blocking and does not affect any pending transport provider data transfer requests. Disassociating resources from a task lets the CPT properly manage resources during task termination. This ability to GIVE and TAKE ownership of connections offers a range of programming options, while still providing CPT with resource management capabilities.

The GIVE service requires the application to set the AFM version number and token fields. No other AFM fields are referenced.

Note: When a connection is established there are internal CPT resources associated with that connection. CPT is responsible for proper clean-up of those resources on task or transaction termination. These resources include storage allocated by CPT, the API, and the transport provider storage.

A server application is a good example of how the GIVE service benefits a user application. A listening task issues the GIVE service and starts a new transaction to handle data transfer. The data transfer transaction then TAKEs the connection. This sequence prevents a connection from being closed (implicitly by the CPT task termination exit) if the server application terminates. However, if the data transfer transaction terminates without issuing an explicit close (CPT CLOSE service), an implicit close is scheduled, and resource management is handled by the CPT task termination exit.

The version number (AFMVERS) indicates the CPT release level in which this user application program is written. This required field is set to a binary 2 and is validated by the GIVE service before processing the request.

The function code (AFMFUNC) indicates the CPT callable service ID. The field is not initialized by a user application program and has little value to the application except for dump analysis. The function code can identify and map an argument list with the error or trace log and dump analysis.

The token (AFMTOKEN) indicates the connection and internal resources to be processed by the GIVE service. This is a required field and is validated by the GIVE service before processing the request.

The AFMOPTCD field specifies GIVE service processing control options, and provides a mechanism for event notification on return to the application program. Currently, this field is not used by application programs.

024040-200100

Example:

This example establishes a server data transfer connection, issues the GIVE service, and starts a data processing transaction. The token is loaded from the ACM and is used by the GIVE service. The return code is checked to determine GIVE service completion status.

```
SAMP1: PROCEDURE OPTIONS (MAIN);

DCL (T09FLST,T09FGIV)
     ENTRY OPTIONS (INTER ASSEMBLER);
DCL (CPTPRT,MSGPRT) POINTER;

DCL 1  CPT_ACM,
           %INCLUDE SYSLIB(T09KPACM);
DCL 1  CPT_AFM,
           %INCLUDE SYSLIB(T09KPAFM);
DCL 1  CPT_RCS,
           %INCLUDE SYSLIB(T09KPRCS);

DCL LISTENING BIT (1) INTI ('1'B);

%INCLUDE T09KPCON;
    DO WHILE (LISTENING);
    /*
       CPT LISTEN Connection Management service request
     */
       AFMTOKEN = ACMTOKEN;
    /*
       CPT GIVE Facility Management service request
     */
       CALL T09FGIV (CPT_AFM);
       IF AFMRTNCD ^= 0
       THEN DO;
       /*
          Process and log GIVE error
          Terminate WHILE condition
        */
       END;
    /*
       START Data Processing Transaction
     */
       CICS START TRANSID(trans id) FROM(CPT_ACM) LENGTH(STG(CPT_ACM));
    END;
    /*
       CPT LISTEN Connection Release service request
     */
    /*
       Terminate transaction
     */
       EXEC   CICS RETURN;
```

200801-024040-200100

LISTEN

The LISTEN service provides a server facility for use by an application program. The LISTEN service establishes a session with the local transport provider, passively listens for connection requests, and then accepts new connections. When connection with a client is established, the LISTEN service either returns control to the calling program or starts a defined transaction. Information related to the connection is updated and returned within the ACM.

To invoke the LISTEN service, a user application is required to first build an ACM and then to issue a call to the LISTEN routine. The minimum information required by this service is version number and either the local transport provider port or service name. Optional information related to data transfer buffering, CPT statistics and tracing, and subtask initialization can be specified. Completion of a LISTEN service depends on options selected within the ACM.

This table describes the LISTEN service arguments:

INCLUDED STATEMENT	SIZE	CREATED BY
T09KPACM ACM	676 (X'2A4')	User application or common area obtained by a RETREIVE command from a started transaction.

Structure

This is the structure of the LISTEN service:

```
DCL 1 CPT_ACM,
/*BEGIN %INCLUDE SYSLIB
(T09KPACM)**********************************
/*************************************************************/
/*                          CPT_ACM                          */
/*     CICS PROGRAMMER'S TOOLKIT CONNECTION MANAGEMENT BLOCK
/*-----------------------------------------------------------*/
5  ACMVERS    FIXED BIN (15) INIT(2),    /* ACM BLOCK VERSION  */
5  ACMFUNC    FIXED BIN (15) INIT(0),    /* FUNCTION CODE      */
5  ACMTOKEN   POINTER  INIT(NULL),       /* DATA XFER TOKEN    */
5  ACMRSVD1   POINTER  INIT(NULL),       /* (RESERVED: IN USE) */
5  ACMRSVD2   FIXED BIN (31) INIT(0),    /* (RESERVED: IN USE) */
5  ACMRTNCD   FIXED BIN (31) INIT(0),    /* RETURN CODE        */
5  ACMDGNCD   FIXED BIN (31) INIT(0),    /* DIAGNOSTIC CODE    */
5  ACMSFILL   CHAR  (3) INIT(LOW(3)),    /* RESERVED STATS     */
5  ACMSTATS   BIT (8) INIT('00000000'B), /* STATISTICS REQS    */
5  ACMTFILL   CHAR  (2) INIT(LOW(2)),    /* RESERVED TRACE     */
5  ACMTRAC2   BIT (8) INIT('00000000'B), /* TRACE OPTIONS 2    */
5  ACMTRAC1   BIT (8) INIT('00000000'B), /* TRACE OPTIONS 1    */
5  ACMQSEND   FIXED BIN (31) INIT(0),    /* SEND QUEUE SIZE    */
5  ACMMSEND   FIXED BIN (31) INIT(0),    /* SEND MAX BUFFER    */
5  ACMQRECV   FIXED BIN (31) INIT(0),    /* RECV QUEUE SIZE    */
5  ACMMRECV   FIXED BIN (31) INIT(0),    /* RECV MAX BUFFER    */
5  ACMTLSTN   FIXED BIN (31) INIT(0),    /* LISTEN TOKEN       */
5  ACMUCNTX   FIXED BIN (31) INIT(0),    /* USER CONTEXT FIELD */
5  ACMTRNID   CHAR   (4) INIT('  '),     /* SERVER TRANSID FOR */
                                         /* LISTEN TO START    */
```

```
5  ACMRSVD4   CHAR   (1) INIT(LOW(1)),   /* (RESERVED: IN USE) */
5  ACMRSVD5   CHAR   (3) INIT(LOW(3)),   /*                    */
5  ACMLPORT   FIXED BIN (15) INIT(0),    /* LOCAL PORT NO.     */
5  ACMRPORT   FIXED BIN (15) INIT(0),    /* REMOTE PORT NO.    */
5  ACMSRVCE   CHAR  (36) INIT('  '),     /* SERVICE NAME       */
5  ACMRSVD6   CHAR   (1) INIT(LOW(1)),   /* (RESERVED: IN USE) */
5  ACMRSVD7   CHAR   (1) INIT(LOW(1)),   /*                    */
5  ACMOPTNS   FIXED BIN (15) INIT(0),    /* OPTION CODES       */
5  ACMLADDR   FIXED BIN (31) INIT(0),    /* LOCAL HOST ADDRESS */
5  ACMRADDR   FIXED BIN (31) INIT(0),    /* REMOTE HOST ADDR   */
5  ACMLNAME   CHAR (255) INIT('  '),     /* LOCAL HOST NAME    */
5  ACMRSVD8   CHAR   (1) INIT(LOW(1)),   /* (RESERVED: IN USE) */
5  ACMRNAME   CHAR (255) INIT('  '),     /* REMOTE HOST NAME   */
5  ACMRSVD9   CHAR   (1) INIT(LOW(1)),   /* (RESERVED: IN USE) */
5  ACMRSVDA   FIXED BIN (15) INIT(0),    /* (RESERVED: IN USE) */
5  ACMRSVDB   FIXED BIN (15) INIT(0),    /* (RESERVED: IN USE) */
5  ACMTIMEO   FIXED BIN (31) INIT(0),    /* TIMEOUT VALUE      */
5  ACMRSVDC   FIXED BIN (31) INIT(0);    /* RESERVED           */

/*END   %INCLUDE SYSLIB
(T09KPACM)**********************************
```

The following constants are contained in the `T09MAC` library member, `T09KPCON`:

```
        /*-------------------------------------------*/
          /*         STATISTICS LOG REQUESTS         */
        /*-------------------------------------------*/
ACMSTATS_CONN BIT (8) INIT('00000001'B), /*   CONNECTIONS   */
ACMSTATS_TERM BIT (8) INIT('00000010'B), /* TRMNATION STATS */

        /*-------------------------------------------*/
          /*            TRACE LOG REQUESTS           */
        /*-------------------------------------------*/
ACMTRAC1_NTRY BIT (8) INIT('00000001'B), /*   ENTRY POINTS  */
ACMTRAC1_ARGS BIT (8) INIT('00000010'B), /*    ARGUMENTS    */
ACMTRAC1_RECV BIT (8) INIT('00000100'B), /*      TRECV      */
ACMTRAC1_SEND BIT (8) INIT('00001000'B), /*      TSEND      */
ACMTRAC1_TERM BIT (8) INIT('00010000'B), /*   TERMINATION   */
ACMTRAC1_PASS BIT (8) INIT('00100000'B), /*    GIVE/TAKE    */
ACMTRAC1_CLSE BIT (8) INIT('01000000'B), /*      CLOSE      */
ACMTRAC1_TERR BIT (8) INIT('10000000'B), /*    TPLERRORS    */
ACMTRAC2_TOKN BIT (8) INIT('00000001'B), /*      TOKENS     */
ACMTRAC2_TPL  BIT (8) INIT('00000010'B), /*        TPL      */
ACMTRAC2_RLSE BIT (8) INIT('00000100'B), /*     RELEASE     */
ACMTRAC2_STOR BIT (8) INIT('00001000'B), /*     STORAGE     */
ACMTRAC2_CLTD BIT (8) INIT('00010000'B), /*   CLIENT-DATA   */

        /*-------------------------------------------*/
          /*              ACM OPTIONS                */
        /*-------------------------------------------*/
ACMOPTN1_SYNC  BIT (8) INIT('00000001'B), /* LISTEN SYNCPOINT*/
ACMOPTN1_LTRAN BIT (8) INIT('00000010'B), /* DYNAM TRANSACTN */
ACMOPTN1_NODNR BIT (8) INIT('00000100'B), /* NO DNR NAMES    */
```

200801-024040-200100

PARAMETER	DESCRIPTION
ACMVERS	Version Indicates the version number of the CPT argument list used by the calling program. This required field is set to a binary 2 for this release of CPT. Default: None
ACMFUNC	Function code Indicates the function or callable service ID requested by the application program. This field is not set by the application, but is initialized by the TRUE *i*nterface stub program. Default: None (generated by service stub)
ACMTOKEN	Data transfer token Specifies the token is created and returned by the LISTEN service. This token is used for all subsequent service calls for the client connection. Default: 0 (token returned)
ACMRTNCD	Return code Indicates the return code set by the LISTEN service. This value is returned and indicates the success or failure of the service. Default: 0
ACMDGNCD	Diagnostic code Indicates the diagnostic code received by the LISTEN service for a transport provider request. A detailed explanation of this value can be found in the transport provider's ***API Programmer's Reference Guide***. Default: 0
ACMSTATS	Specifies statistics logging options for the application program. This facility can be used for debugging and tuning during development. ◆ ACMSTATS_CONN – Specifies that a message(s) be generated when either a listen service or a data transfer connection is established. These messages are generated by the CPT LISTEN and CONNECT services. The message numbers associated with this option are CPT802I and CPT803I. ◆ ACMSTATS_TERM – Specifies that a message(s) be generated on terminating an established connection. These messages are generated by the CPT CLOSE service. The message numbers associated with this option are CPT807I, CPT808I, and CPT809I. Default: 0 (no statistics logging)

PARAMETER	DESCRIPTION
ACMTRAC1	Specifies trace logging options for the application program. This facility can be used for debugging during development. ◆ ACMTRAC1_NTRY – Specifies that a message be generated on entry to a CPT service routine. Messages are generated by all CPT service routines. The message number associated with this option is CPT901I. ◆ ACMTRAC1_ARGS – Specifies that a hex dump of the caller's arguments be generated on entry and exit of a CPT service routine. Messages are generated by the corresponding service routine. The message numbers associated with this option are CPT902I, CPT903I, CPT904I, CPT911I, CPT912, CPT921I, CPT922I, CPT924I, CPT925I, CPT930I, and CPT932I. ◆ ACMTRAC1_RECV – Specifies that a hex dump of the transport provider's input (RECEIVE) data be logged. Messages are generated by the CPT RECV service. The message number associated with this option is CPT913I. ◆ ACMTRAC1_SEND – Specifies that a hex dump of the transport provider's output (SEND) data be logged. Messages are generated by the CPT SEND service. The message number associated with this option is CPT914I. ◆ ACMTRAC1_TERM – Specifies that a message be generated on termination of a CPT service routine. Messages are generated by all CPT service routines. The message number associated with this option is CPT908I. ◆ ACMTRAC1_PASS – Specifies that a hex dump of resources related to a passed connection be logged. Messages are generated by the CPT GIVE and TAKE services. The message numbers associated with this option are CPT919I and CPT920I. ◆ ACMTRAC1_CLSE – Specifies that a hex dump of resources related to a CLOSE processing of a connection be logged. Messages are generated by the CPT CLOSE service. The message number associated with this option is CPT906I ◆ ACMTRAC1_TERR – Specifies that a hex dump of a transport provider API parameter list that fails successful completion be logged. The message number associated with this option is CPT400I.

200801-024040-200100

PARAMETER	DESCRIPTION
ACMTRAC2	◆ ACMTRAC2_TOKN – Specifies that a hex dump of the connection token be logged. Messages are generated by all services routines on entry. The message number associated with this option is CPT909I. ◆ ACMTRAC2_TPL – Specifies that a hex dump of the transport provider API parameter list be logged. The message numbers associated with this option are CPT917I and CTP935I. ◆ ACMTRAC2_RLSE – Specifies that a message indicating a transport provider release indication is to be logged. Messages are generated by various CPT service routines. The message number associated with this option is CPT414I. ◆ ACMTRAC2_STOR – Specifies that a hex dump of storage management argument to be logged. Messages are generated by various CPT service routines. The message numbers associated with this option are CPT928I and CPT929I. ◆ ACMTRACE2_CLTD – Trace transient data writes from the LISTEN service (used with the ACMOPTNS_LTRAN client-data option). The message number associated with this option is CPT918I. Default: 0 (no trace logging)
ACMQSEND	API send queue size Specifies the maximum number of uncompleted SEND requests that can be queued by the application to the transport provider (API). This value lets applications control output processing and can affect throughput rates. The value is negotiated with and can be modified by the transport provider. The total allocation for output processing is the product of the SEND queue and buffer size values and cannot exceed 61K. Default: 4
ACMMSEND	API send buffer size Specifies the maximum number of user data bytes that can be transferred by the application in a single SEND request to the transport provider (API). This value lets applications control output processing and can affect throughput rates. The value is negotiated with and can be modified by the transport provider. The total allocation for output processing is the product of the SEND queue and buffer size values and cannot exceed 61K. Default: 4096
ACMQRECV	API receive queue size Specifies the maximum number of uncompleted RECEIVE requests that can be queued by the application to the transport provider (API). This value lets applications control input processing and can affect throughput rates. The value is negotiated with and can be modified by the transport provider. The total allocation for input processing is the product of the RECEIVE queue and buffer size values and cannot exceed 61K. Default: 4
ACMMRECV	API receive buffer size Specifies the maximum number of user data bytes that can be transferred by the application, in a single RECEIVE request, to the transport provider (API). This value lets applications control input processing and can affect throughput rates. The value is negotiated with and can be modified by the transport provider. The total allocation for input processing is the product of the RECEIVE queue and buffer size values and cannot exceed 61K. Default: 1024

2024040-200100

PARAMETER	DESCRIPTION
ACMTLSTN	Listen service token statistics Specifies the token used by the LISTEN service. This token is not available for data transfer. The only valid function that can be performed is a CLOSE request for long running active listeners. Generally, this value is not used by the application unless an explicit call to the CLOSE service is required. Read the description for ACMTOKEN (earlier in this table) for all other services. Default: 0 (token returned)
ACMUCNTX	One word of user context Specifies one arbitrary word of user context to be associated with the connection. The information provided is not interpreted by CPT, and is merely saved with other connection information. Default: 0 (no user context)
ACMTRNID	Listen start transaction ID This field is not used by the CONNECT service. The value in this field is not validated nor is it modified. Default: None
ACMLPORT	Listen well-known service port Indicates the local transport layer address or port. This value represents the well-known port on which a server application will listen for connection requests. Either this value or the transport layer service name (ACMSRVCE) must be specified. This field is an unsigned positive integer with a maximum value of 65,534. The value must be unique for each server application. Default: None
ACMRPORT	Remote well-known service port Indicates the remote transport layer address or port. This value represents the client port number. This value is returned to the caller. This field is an unsigned positive integer with a maximum value of 65,534. Default: None
ACMSRVCE	Transport layer service name Indicates the local transport layer service name. This string is mapped into a transport layer address or port, through internal DNR. The resolved value is the well-known port to which the client application connects. This field has a maximum length of 36 bytes. The value must be unique for each server application. This field is optional and is not modified by the LISTEN service. Default: None
ACMOPTNS	TCP connection initialization options ◆ ACMOPTN1_NODNR – DNR Suppression option. Skips internal DNR calls to resolve and return the remote IP address into an IP name in the ACMRNAME field. If an application is designed such that TCP connection establishment and release happens frequently, this option can save processing time. ◆ ACMOPTN1_LTRAN – Client-Data Listener option. Specifies that the Listen call will receive the input datastream to determine the transaction ID to be started. See Client-Data Listener options for the required input formats. This option must be used with ACMTIMEO, and should not be used with ACMTRANID. ◆ ACMOPTN1_SYNC – Listen Syncpoint option. Issues a CICS syncpoint before starting any transaction from the LISTEN service. Default: None

200R01-024040-200100

PARAMETER	DESCRIPTION
ACMLADDR	Local IP host address Indicates the local host internet address. This field is an unsigned four-byte integer value. The local host internet address is updated when a server connection is established, and is returned to the caller. Default: None
ACMRADDR	Remote IP host address Indicates the remote host internet address. Either this field or the remote host name (ACMRNAME) field must be specified. This field is an unsigned four-byte integer value. The remote host internet address is updated when a server connection is established, and is returned to the caller. Default: None
ACMLNAME	Local IP host name Indicates the local host internet name. This field is a 255-byte character string that is padded with blanks. The local host internet name is updated when a client connection is established, and is returned to the caller. Default: None
ACMRNAME	Remote IP host name Indicates the remote host internet name. Either this value or the remote IP address (ACMRADDR) field must be specified. This field is a 255-byte character string that is padded with blanks. The remote host internet name is updated when a server connection is established, and is returned to the caller. Default: None
ACMTIMEO	Client-Data Listener timeout values Specifies the maximum number of seconds that a Listener can wait to receive the client datastream when the ACMOPTN1_LTRAN option is specified. Default: 1 second

Network Considerations

The ACM is a common data structure used for both client and server connection initialization. There are common and unique values specified for a particular service request.

This table describes network considerations for PL/1 API:

NAME	SERVER CONDITIONS FOR LISTEN	CLIENT CONDITIONS FOR CONNECT
ACMTLSTN	Listen token returned to user application.	
ACMTRNID	Listen START transaction ID.	
ACMLPORT	Local server or listening transport provider well-known port selected by user application.	Local assigned transport provider port returned to user application.
ACMRPORT	Remote client transport provider port returned to user application.	Remote server transport provider well-known port selected by user application.

2024040-200100

NAME	SERVER CONDITIONS FOR LISTEN	CLIENT CONDITIONS FOR CONNECT
ACMSRVCE	Local server transport provider service name selected by user application.	Remote server transport provider service name selected by user application.
ACMRADDR	Remote IP host address returned to user application.	Remote IP host address selected or returned to user application.
ACMLNAME	Local IP host name returned to user application.	Remote IP host name returned to user application.
ACMRNAME	Remote IP host name returned to user application.	Remote IP host name selected or returned to user application.
ACMTIMEO	Client-Data Listener timeout value.	

200801-024040-200100

Completion Information

Completion of a request to the LISTEN service depends on the arguments selected. If no transaction ID is specified, the LISTEN service returns control to the calling program when a connection with a client is established. The caller's argument list is updated with information related to the new connection. If a transaction ID is specified, the LISTEN service does not return control to the calling program until a failure is detected. The caller's argument list is generally not updated, except for the return code information.

The LISTEN service initializes the server environment with the transport provider (API), waits for a connection request, establishes a connection with the client, and updates connection information within the ACM. Establishing a listening connection and a client connection are represented by storage and are referred to hereafter as tokens. Establishing a client connection updates the ACM with information relative to the connection. The information is returned to the user or is passed to the data processing transaction.

The server application contains two tokens representing endpoints to the transport provider. The first token (ACMTOKEN) represents the client connection and is used for data transfer. The other token (ACMTLSTN) represents the listening connection. This listening connection can only be referenced within the CPT CLOSE service. This lets an explicit ability close a server or listening connection. All other CPT services performed with the LISTEN token fail with an invalid token. Implicit clean-up of the LISTEN token is provided by the TRUE interface. Therefore, an explicit call to the CLOSE service is not required.

When the LISTEN service is initiated without a transaction ID, control is returned to the calling program when a connection with a client is established. The caller's argument list is updated with information related to the new connection. The local and remote port, IP address, and host names are resolved. Negotiated transport provider SEND and RECEIVE buffering values are returned. The ACM return code ACMRTNCD must be checked to determine the success or failure of LISTEN service. A zero (0) return code indicates a successful client connection is established.

When the LISTEN service is initiated with a transaction ID, it operates as a CICS long running task. The LISTEN service establishes client connections and starts a data processing transaction. The data processing transaction receives a copy of the connection management argument. The data transfer or client connection token is derived from the ACMTOKEN field. After the new transaction is initiated the LISTEN service continues waiting for new client connections. The LISTEN service continues to listen and start client connections until an error occurs.

The return and diagnostic codes should be interrogated to determine the reason for failure. Errors indicating CPT, the transport provider (API), or CICS termination are minor. Errors should be interrogated for their level of severity.

Return Codes

The LISTEN service return and diagnostic codes indicate the results of the execution. These values are in the ACMRTNCD and ACMDGNCD within the ACM. The diagnostic code is optional and indicates the transport provider return code. Read **Appendix C – MESSAGES AND CODES** in the ***CICS Programmer's Toolkit Installation and Administration Guide*** for the return code cross reference table.

This table describes the LISTEN service return codes:

RETURN	DIAGNOSTIC CODE	DESCRIPTION
CPTIRCOK		Successful.
CPTEVERS		Control block version number is not supported.
CPTECONN	Yes	Requested host/service/port is not found.
CPTEPROT		Specified protocol is not supported.
CPTENAPI	Yes	Transport provider API is not available.
CPTETERM	Yes	Environment is being terminated.
CPTERLSE	Yes	Release indication.
CPTEDISC	Yes	Disconnect indication.
CPTEPRGE	Yes	CPT Interface terminating.
CPTEINTG	Yes	Transport provider API integrity error.
CPTEENVR	Yes	Transport provider API environment error.
CPTEFRMT	Yes	Transport provider API format error.
CPTEPROC	Yes	Transport provider API procedure error.
CPTABEND		Abnormal exception occurred.
CPTEOTHR	Yes	An undefined exception occurred.

200801-024040-200100

Usage Information

The LISTEN service lets user-written application programs implement TCP/IP server facilities. Server applications passively wait, then establish connections with single- or multi-thread support. The LISTEN service's generalized parameter list (ACM) describes the application's communications requirements as well as information related to established connections. The ACM contains fields initialized by both a user application and by the LISTEN service, on completion.

There are required and optional fields initialized by a user or calling application. The ACM version number and the transport provider local port or service name are required. The selection of port or service name defines the server well-known port address. Optional fields control data transfer buffering, statistics, tracing, and subtask initialization.

When the LISTEN service completes or the data processing task executes, the ACM will contain information related to the established connection. A token that identifies the connection is returned in the ACM, and must be used in all subsequent requests that refer to the connection. The application program should make no assumptions regarding the format of a token, other than it is an unsigned, full word value.

Information related to the negotiated buffer values, host names, host addresses, and transport provider addresses are returned in the ACM.

The version number (ACMVERS) indicates the CPT release level in which this user application program is written. This required field is set to a binary 2 and is validated by the LISTEN service before it processes the request.

The function code (ACMFUNC) indicates the CPT callable service ID. The field is initialized by the CPT service stub program. The function code identifies argument lists within the error or trace logs, and dump analysis.

The transport provider port number (ACMPORT) or service name (ACMSRVCE) is required. These fields identify the well-known port to which a client initiates a connection request. The port number has precedence over service name. This implies that the service name field is only used if a port number is not specified.

The transaction ID field (ACMTRNID) identifies a data processing task. This is an optional field that causes the LISTEN service to execute continuously. The LISTEN service starts a new transaction after a client connection is established, then waits for additional connection requests. A updated ACM is passed to the data processing task. Control is not returned to the calling program until an error occurs. The return code indicates the reason for the failure. Errors indicating the transport provider, CICS, or CPT termination are acceptable. Errors indicating port in use, API unavailable, or program checks should be investigated.

User application programs can control CPT and transport provider data transfer buffering. The ACMQSEND, ACMMSEND, ACMQRECV, and ACMMRECV specify the number and size of buffers allocated. The SEND and RECEIVE buffers are allocated on initial entry into either the SEND or RECEIVE service. The corresponding values used by the SEND and RECEIVE services are independent of each other. The SEND service multiplies the queue and buffer values to determine output storage requirements. The RECEIVE service performs a

similar function to determine input storage requirements. The product of the queue and buffer values cannot exceed 61,440. CPT requires some additional storage to manage these buffers. This extra storage is included in the allocation.

The CPT SEND service uses the ACMQSEND value; the CPT RECEIVE service uses the ACMQRECV value. These values indicate the maximum number of uncompleted SEND or RECEIVE requests that can be queued by the application to the transport provider. The CPT data transfer services schedule asynchronous transport provider or API requests on the caller's behalf. These API requests must be completed before they can be used again. If the queue values are too small the CPT data transfer service may block the caller's request and schedule a wait within the service routines. If the queue values are too large the user application may be wasting storage.

The CPT SEND service uses the ACMMSEND value; the CPT RECEIVE service uses the ACMMRECV value. These values indicate the maximum number of user data bytes that can be transferred by the application in a single SEND or RECEIVE request to the transport provider. The user application is not limited to these values within the data transfer services. However, it is important to note that multiple transport provider or API requests are issued to complete the caller's request. Information on queue and buffer size are in the sections **RECEIVE** on page 5-53 and **SEND** on page 5-63.

Initially, the tuning of data transfer storage may not be a concern. However, the ability to control storage allocation can prove beneficial to the application or CICS region. Queue size can increase data transfer throughput. The application programmer should consider enabling the statistics option to gather CPT statistical information. This information can set the SEND or RECEIVE queue and buffer size values.

The LISTEN service can modify the data transfer buffer allocation values. These values are negotiated with the transport provider and, depending on the site configuration, can be reduced. Any application dependent on these values should check them on return. These values are generally not modified when giving reasonable numbers. However, it is advisable to check with the site administrator for the maximum values of the API transport services.

A number of arguments are not set by the calling application, but are returned to the caller. These values represent information related to the client connection and can be used by the application. The local port, host name, and IP address are returned as well as the client's corresponding values. An ACM is passed the started transaction when a TRANSID is specified in the caller's listen argument list.

200801-024040-200100

Client-Data Listener Option

The option ACMOPTN1_LTRAN is used in conjunction with ACMTIMEO and is mutually exclusive of the use of the ACMTRNID field. ACMOPTN1_LTRAN indicates to the LISTEN service that the connecting client application will specify what server functions to execute. When the LISTEN service receives a CONNECT request and ACMOPTN1_LTRAN is specified, it uses a partial record timed RECEIVE (see RECEIVE service options) to get the client's data. It uses ACMTIMEO to know how long to wait for the client data which can be in any of these formats:

```
TRAN
TRAN,UUUUUUUUUUUU
TRAN,UUUUUUUUUUUU,IC,HHMMSS
TDQN,UUUUUUUUUUUU,TD
TRAN,,IC,HHMMSS
TDQN,,TD
```

PARAMETER	DESCRIPTION
TRAN	A 1- to 4-character transaction ID to start, passing CLNT_PARM to the started transaction
UUUUUUUUUUUU	A 1- to 35-byte user data and is passed to the started transaction or written to the transient data queue in the field CP_DATA.
IC	Specifies that TRAN is to be started in HHMMSS; if left blank, startup is immediate.
HHMMSS	Hours, minutes, and seconds for IC option.
TD	Indicates that CLNT_PARM will be written into the transient data queue, TDQN.

```
/*************************************************************
/*                  PL/I CLNT_PARM DESCRIPTION
/*-----------------------------------------------------------
DECLARE 1 CLNT_PARM,
          3 CP_TOKEN FIXED BIN (31), /*TOKEN*/
          3 CP_FILL1 CHAR      (16),
          3 CP_DATA CHAR       (36), /*CLIENT DATA*/
          3 CP_ADDRESS,
             5 CP_DOMAIN FIXED BIN   (15), /*FAMILY*/
             5 CP_RPORT FIXED BIN    (15), /*REMOTE PORT*/
             5 CP_RADDR FIXED BIN    (15), /*REMOTE IP ADDRESS*/
          3 CP_FILL2 CHAR      (8);
```

Note: Using this option puts a listener at risk of being tied up until the client actually sends the data. Set ACMTIMEO with this fact in mind. The new trace flag, ACMTCLTD, can optionally be specified to trace the CLNT_PARM written to TDQN transient data queue via CPT918I.

Example:

This example establishes a server connection, processes data, and closes the connection. The server listens on well-known port 1234. The token is loaded from the ACM and used by all of the following CPT service requests. The return code is checked to determine `LISTEN` service completion status.

This sample program is generally not the preferred server model. The problem is that after returning from the `LISTEN` service the application blocks additional incoming connection requests. (A better example of this facility is show in the second example.) This single-threaded server model is really only suitable for connections of a very short time duration.

```
SAMP1: PROCEDURE OPTIONS (MAIN);

DCL    (T09FLST)
       ENTRY OPTIONS (INTER ASSEMBLER);
DCL    (CPTPRT,MSGPRT) POINTER;

DCL 1  CPT_ACM,
       %INCLUDE SYSLIB(T09KPACM);
DCL 1  CPT_RCS,
       %INCLUDE SYSLIB(T09KPRCS);

DCL LISTENING BIT (1) INTI ('1'B);

%INCLUDE T09KPCON;

   DO WHILE (LISTENING);
   /*
      CPT LISTEN Connection Management service request
    */
      ACMLPORT = 1234;
      CALL T09FLST (CPT_ACM);
      IF ACMRTNCD ^= 0
      THEN DO;
      /*
         Process and log LISTEN error
         Terminate WHILE condition
       */    END;

      ADTTOKEN, ACLTOKEN = ACMTOKEN;

   /*
      Application and CPT Data Transfer (SEND/RECV) service request
    */

   /*
      CPT Data Transfer Connection Release service request
    */

   END;

   /*
      Terminate transaction
    */

      EXEC   CICS RETURN;
```

200801-024040-200100

Example:

This example is a multi-threaded server application. This task initiates client connections and then starts a data processing transaction. The server listens on the port mapped into service name ECHO. The token is loaded from the ACM and then is passed to the data transfer transaction. The return code is checked to determine LISTEN service completion status.

This sample program differs from the previous example in that data transfer is not performed by the listening transaction, but by a different transaction. This allows for a more efficient server program. The server application is better able to respond quickly to new connection requests, because it is not involved in the tedious task of data transfer or connection management after the initialization connection.

This example does not use the optional GIVE Facility Management service. The GIVE service is beneficial if this task was expected to terminate before the data processing transaction initiated. However, since this task is not expected to terminate, any abnormal termination releases all CPT connections currently associated with this task.

```
SAMP2: PROCEDURE OPTIONS (MAIN);

DCL    (T09FLST,T09FGIV)
       ENTRY OPTIONS (INTER ASSEMBLER);
DCL    (CPTPRT,MSGPRT) POINTER;

DCL 1  CPT_ACM,
       %INCLUDE SYSLIB(T09KPACM);
DCL 1  CPT_AFM,
       %INCLUDE SYSLIB(T09KPAFM);
DCL 1  CPT_RCS,
       %INCLUDE SYSLIB(T09KPRCS);

DCL LISTENING BIT (1) INTI ('1'B);

%INCLUDE T09KPCON;

   DO WHILE (LISTENING);
   /*
      CPT LISTEN Connection Management service request
    */
      ACMSRVCE = 'ECHO';

   CALL T09FLST (CPT_ACM);
      IF ACMRTNCD ^= 0
      THEN DO;/*
         Process and log LISTEN error
         Terminate WHILE condition
       */
      END;

      AFMTOKEN = ACMTOKEN;

      /*
         START Data Processing Transaction
       */
         CICS  START TRANSID(trans id)
             FROM(CPT_ACM) LENGTH(STG(CPT_ACM));

   END;
```

Example:

In this example a multi-threaded server listens for connections resolved to the transport service name DISCARD. A TRANSID is specified that causes the LISTEN service to start a data processing transaction. Control is not returned to this program until an error occurs. The ACMRTNCD is checked to determine LISTEN service request completion status.

This sample program differs from the second example in that a START command for a new transaction is performed by the LISTEN service and not by the user application. Also, control is not returned to the calling application until a failure occurs. Generally, this failure is due to termination of CICS, CPT, or the transport provider (API).

In this example, transaction SRV3 is automatically started by the LISTEN service for each connection established on the DISCARD port.

```
SAMP2: PROCEDURE OPTIONS (MAIN);

DCL    (T09FLST)
       ENTRY OPTIONS (INTER ASSEMBLER);
DCL    (CPTPRT,MSGPRT) POINTER;

DCL 1  CPT_ACM,
       %INCLUDE SYSLIB(T09KPACM);
DCL 1  CPT_RCS,
       %INCLUDE SYSLIB(T09KPRCS);

%INCLUDE T09KPCON;

   /*
      CPT LISTEN Connection Management service request
    */
      ACMSRVCE = 'DISCARD';
      ACMTRNID = 'SRV3';
      CALL T09FLST (CPT_ACM);
      IF ACMRTNCD ^= 0
      THEN DO;
      /*
         Process and log LISTEN error
       */
      END;

   /*
      Terminate transaction
    */
      EXEC   CICS RETURN;
```

200801-024040-200100

RCVFROM

The RCVFROM service lets you develop connectionless client and server applications. This service is UDP only. The RCVFROM service provides these basic functions:

- Establishes a UDP server endpoint represented by a new token and starts receiving datagrams on a user-specified well-known port. Indicate this function to the RCVFROM service by passing an ADTTOKEN equal to zero. RCVFROM then creates all the internal control blocks and the RCVFROM buffer queue. Even though the SENDTO buffer queue is not allocated for this endpoint (token) until the SENDTO service is called, the SENDTO buffer size and number must be specified at this time because they are negotiated with the transport provider and recorded in the internal CPT control blocks at endpoint creation time. Upon return from the RCVFROM service, ADTTOKEN contains the token value to be passed to subsequent RCVFROM and SENDTO service calls.
- Receives a datagram at a previously established UDP endpoint represented by an existing token. This functionality makes the RCVFROM service call just a data transfer call that can be used by a client or server application. The RCVFROM buffer queue is only allocated upon the first call to the RCVFROM service, whether or not ADTTOKEN is equal to zero.

UDP tokens created with the RCVFROM or SENDTO services cannot be passed to the TCP-only services, CONNECT, LISTEN, SEND, and RECEIVE. All other CPT service calls are available to UDP applications.

The non-blocking option of the RCVFROM service (ADTOPCD1=ADTNBLKR), allows applications to be developed that can poll a well-known UDP port or send to a remote UDP server and then make a predetermined number of RCVFROM calls to get back a response. Given the general unreliable nature of UDP, not blocking on a RCVFROM call can build in some flexibility with regards to handling lost datagrams.

This table describes the RECVFROM service arguments:

INCLUDE STATEMENT	SIZE	CREATED BY
T09KPADT	644 (X'284')	User application

Structure

This is the structure of the RECVFROM service in PL/1 language:

```
DCL 1 CPT_ADT,
/*BEGIN %INCLUDE SYSLIB
(T09KPADT)**********************************
/*************************************************************/
/*                          CPT_ADT                          */
/*         CICS PROGRAMMER'S TOOLKIT DATA TRANSFER BLOCK */
/*-----------------------------------------------------------*/
5  ADTVERS    FIXED BIN (15) INIT(2),    /* ADT BLOCK VERSION  */
5  ADTFUNC    FIXED BIN (15) INIT(0),    /* FUNCTION CODE      */
5  ADTTOKEN   POINTER  INIT(NULL),       /* DATA XFER TOKEN    */
5  ADTBUFFA   POINTER  INIT(NULL),       /* DATA BUFFER ADDRSS */
5  ADTBUFFL   FIXED BIN (31) INIT(0),    /* BUFFER/DATA LENGTH */
5  ADTRTNCD   FIXED BIN (31) INIT(0),    /* RETURN CODE        */
5  ADTDGNCD   FIXED BIN (31) INIT(0),    /* DIAGNOSTIC CODE    */
5  ADTSFILL   CHAR  (3) INIT(LOW(3)),    /* RESERVED STATS     */
5  ADTSTATS   BIT (8) INIT('00000000'B), /* STATISTICS REQS    */
5  ADTTFILL   CHAR  (2) INIT(LOW(2)),    /* RESERVED TRACE     */
5  ADTTRAC2   BIT (8) INIT('00000000'B), /* TRACE OPTIONS 2    */
5  ADTTRAC1   BIT (8) INIT('00000000'B), /* TRACE OPTIONS 1    */
5  ADTQSEND   FIXED BIN (31) INIT(0),    /* TSEND QUEUE SIZE   */
5  ADTMSEND   FIXED BIN (31) INIT(0),    /* TSEND BUFFER SIZE  */
5  ADTQRECV   FIXED BIN (31) INIT(0),    /* TRECV QUEUE SIZE   */
5  ADTMRECV   FIXED BIN (31) INIT(0),    /* TRECV BUFFER SIZE  */
5  ADTTIMEO   FIXED BIN (31) INIT(0),    /* TIMED RECV SECONDS */
5  ADTRSVD1   FIXED BIN (31) INIT(0),    /*      RESERVED      */
5  ADTRSVD2   FIXED BIN (31) INIT(0),    /*      RESERVED      */
5  ADTRSVD3   FIXED BIN (31) INIT(0),    /*      RESERVED      */
5  ADTRSVD4   FIXED BIN (31) INIT(0),    /*      RESERVED      */
5  ADTLPORT   FIXED BIN (15) INIT(0),    /* LOCAL PORT         */
5  ADTRPORT   FIXED BIN (15) INIT(0),    /* REMOTE PORT        */
5  ADTRSVD5   FIXED BIN (15) INIT(0),    /*      RESERVED      */
5  ADTSRVCE       CHAR (36) INIT(' '),  /* SERVICE NAME       */
5  ADTSEP#    FIXED BIN (15) INIT(0),    /* # OF SEP CHARS     */
5  ADTSEP1        CHAR  (1) INIT(' '),  /* 1ST/ONLY SEP CHAR  */
5  ADTSEP2        CHAR  (1) INIT(' '),  /* 2ND SEP CHARACTER  */
5  ADTRSVD6   FIXED BIN (15) INIT(0),    /*     RESERVED      */
5  ADTLADDR   FIXED BIN (31) INIT(0),    /* LOCAL HOST IP ADDR */
5  ADTRADDR   FIXED BIN (31) INIT(0),    /* REMOTE HOST IP ADR */
5  ADTLNAME      CHAR (255) INIT(' '),  /* LOCAL HOST NAME    */
5  ADTRSVD7       CHAR  (1) INIT(' '),  /*     RESERVED       */
5  ADTRNAME      CHAR (255) INIT(' '),  /* REMOTE HOST NAME   */
5  ADTRSVD8       CHAR  (1) INIT(' '),  /*     RESERVED       */
5  ADTUCNTX   FIXED BIN (31) INIT(0),    /* USER CONTEXT FIELD */
5  ADTOPTN4        BIT  (8) INIT('0'B), /*     RESERVED       */
5  ADTOPTN3        BIT  (8) INIT('0'B), /*     RESERVED       */
5  ADTOPTN2        BIT  (8) INIT('0'B), /* OPTIONS - BYTE 2   */
5  ADTOPTN1        BIT  (8) INIT('0'B); /* OPTIONS - BYTE 1   */
/*END   %INCLUDE SYSLIB
(T09KPADT)**********************************
```

The following constants are contained in the T09MAC library member, T09KPCON:

```
/*-------------------------------------------------------------
/*  ADTSTATS AND ADTTRACE VALUES ARE THE SAME AS DEFINED FOR   */
/*  THE ACM.  THESE FIELDS CAN THUS BE SET TO ACM-CONSTANTS.   */
/*-------------------------------------------------------------*/

        /*-------------------------------------------*/
           /*              ADT OPTIONS                */
        /*-------------------------------------------*/
ADTOPTN1_TYPSP BIT (8) INIT('00000001'B), /* RECD BY SEP CHAR*/
ADTOPTN1_TYPLL BIT (8) INIT('00000010'B), /* RECD BY LL PFX  */
ADTOPTN1_BLCKS BIT (8) INIT('00000100'B), /* BLOCKING SEND   */
ADTOPTN1_TMPRT BIT (8) INIT('00001000'B), /* TIMED PART RECV */
ADTOPTN1_TMRCV BIT (8) INIT('00010000'B), /* TIMED FULL RECV */
ADTOPTN1_NBLKR BIT (8) INIT('01000000'B), /* NO BLOCK RECV   */
ADTOPTN1_DODNR BIT (8) INIT('10000000'B), /* DO DNR FOR UDP  */
ADTOPTN2_NOSTP BIT (8) INIT('01000000'B), /* NO STRIP LL/SEP */
ADTOPTN2_VLIST BIT (8) INIT('10000000'B), /* VECTOR LIST     */
```

PARAMETER	DESCRIPTION
ADTVERS	Version Indicates the CPT version number of the argument used by the calling program. This required field is set to a binary 2 for this release of CPT. Default: None
ADTFUNC	Function code Indicates the function or callable service ID requested by the application program. This field is not set by the application, but is initialized by the TRUE interface stub program. Default: None
ADTTOKEN	Data transfer token Specifies a token that represents a UDP endpoint. If the ADT is being passed in a call to either the RCVFROM or SENDTO service, then the token can be zero, indicating to either service, to first create a token before sending or receiving a datagram. If the token is not zero, then it must be a token created previously by either the RCVFROM or SENDTO service. It is not necessary or efficient to create a token every time a CICS transaction calls the UDP data transfer services. It is an error to pass a TCP token to the UDP data transfer service routines, RCVFROM and SENDTO. Conversely it is an error to pass a UDP token to the TCP data transfer routines, RECEIVE and SEND. Default: 0
ADTBUFFA	User data address Indicates the storage address into which the UDP datagram is received (RCVFROM service). This is a contiguous segment of storage accessible to the user task. The content of all user data is application dependent, and not interpreted by either CPT nor the transport provider. The storage area can be aligned on any boundary convenient for the application program. Default: 0
ADTBUFFL	User data length Indicates the length (in bytes) of the buffer specified in ADTBUFFA which is to receive (RCVFROM service) the UDP datagram. If the incoming datagram does not fit into ADTBUFFA for a length of ADTBUFFL, then the warning, CPTWNEOM will be passed back to the caller in ADTRTNCD, indicating that more RCVFROM calls will be required to get the entire datagram. ADTBUFFL will indicate the actual length returned in ADTBUFFA. It is an error to call the RCVFROM service with an ADTBUFFL of zero.

PARAMETER	DESCRIPTION
ADTRTNCD	Return code Indicates the return code set by the RCVFROM service. This value is returned and indicates the success or failure of the service. Default: 0
ADTDGNCD	Diagnostic code Indicates the diagnostic code set by the service request. This value generally indicates a transport provider return code. Default: 0
ADTSTATS	Specifies statistics logging options for the application program. This facility can be used for debugging and tuning during development. ◆ ADTSTATS_CONN – Specifies that a message(s) be generated when either a listen service or a data transfer connection is established. These messages are generated by the CPT LISTEN and CONNECT services. The message numbers associated with this option are CPT802I and CPT803I. ◆ ADTSTATS_TERM – Specifies that a message(s) be generated on terminating an established connection. These messages are generated by the CPT CLOSE service. The message numbers associated with this option are CPT807I, CPT808I, and CPT809I. Default: 0 (no statistics logging)
ADTTRAC1	Specifies trace logging options for the application program. This facility can be used for debugging during development. ◆ ADTTRAC1_NTRY – Specifies that a message be generated on entry to a CPT service routine. Messages are generated by all CPT service routines. The message number associated with this option is CPT901I. ◆ ADTTRAC1_ARGS – Specifies that a hex dump of the caller's arguments be generated on entry and exit of a CPT service routine. Messages are generated by the corresponding service routine. The message numbers associated with this option are CPT902I, CPT903I, CPT904I, CPT911I, CPT912, CPT921I, CPT922I, CPT924I, CPT925I, CPT930I, and CPT932I. ◆ ADTTRAC1_RECV – Specifies that a hex dump of the transport provider's input (RECEIVE) data be logged. Messages are generated by the CPT RECV service. The message number associated with this option is CPT913I. ◆ ADTTRAC1_SEND – Specifies that a hex dump of the transport provider's output (SEND) data be logged. Messages are generated by the CPT SEND service. The message number associated with this option is CPT914I. ◆ ADTTRAC1_TERM – Specifies that a message be generated on termination of a CPT service routine. Messages are generated by all CPT service routines. The message number associated with this option is CPT908I. ◆ ADTTRAC1_PASS – Specifies that a hex dump of resources related to a passed connection be logged. Messages are generated by the CPT GIVE and TAKE services. The message numbers associated with this option are CPT919I and CPT920I. ◆ ADTTRAC1_CLSE – Specifies that a hex dump of resources related to a CLOSE processing of a connection be logged. Messages are generated by the CPT CLOSE service. The message number associated with this option is CPT906I ◆ ADTTRAC1_TERR – Specifies that a hex dump of a transport provider API parameter list that fails successful completion be logged. The message number associated with this option is CPT400I.

PARAMETER	DESCRIPTION
ADTTRAC2	◆ ADTTRAC2_TOKN – Specifies that a hex dump of the connection token be logged. Messages are generated by all services routines on entry. The message number associated with this option is CPT909I. ◆ ACDTTRAC2_TPL – Specifies that a hex dump of the transport provider API parameter list be logged. The message numbers associated with this option are CPT917I and CTP935I. ◆ ADTTRAC2_RLSE – Specifies that a message indicating a transport provider release indication is to be logged. Messages are generated by various CPT service routines. The message number associated with this option is CPT414I. ◆ ADTTRAC2_STOR – Specifies that a hex dump of storage management argument to be logged. Messages are generated by various CPT service routines. The message numbers associated with this option are CPT928I and CPT929I. Default: 0 (no trace logging)
ADTQSEND	API send queue size (used when ADTTOKEN=0) Specifies the maximum number of uncompleted SENDTO requests that can be queued by the application to the transport provider (API). This value lets applications control output processing and can affect throughput rates. The value is negotiated with the transport provider and can be modified by the transport provider. The total allocation for output processing is the product of the SENDTO queue and buffer size values and cannot exceed 61K. Default: 4
ADTMSEND	API send buffer size (used when ADTTOKEN=0) Specifies the maximum number of user data bytes that can be transferred by the application in a single SENDTO request to the transport provider (API). This value lets applications control output processing and can affect throughput rates. The value is negotiated with the transport provider and can be modified by the transport provider. The total allocation for output processing is the product of the SENDTO queue and buffer size values and cannot exceed 61K. Default: 4096
ADTQRECV	API receive queue size (used when ADTTOKEN=0) Specifies maximum number of uncompleted RCVFROM requests that can be queued by the application to the transport provider (API). This value lets applications control input processing and can affect throughput rates. The value is negotiated with the transport provider and can be modified by the transport provider. The total allocation for input processing is the product of the RCVFROM queue and buffer size values and cannot exceed 61K. Default: 4
ADTMRECV	API receive buffer size (used when ADTTOKEN=0) Specifies the maximum number of user data bytes that can be transferred by the application in a single RCVFROM request to the transport provider (API). This value lets applications control input processing and can affect throughput rates. The value is negotiated with the transport provider and can be modified by the transport provider. The total allocation for input processing is the product of the RCVFROM queue and buffer size values and cannot exceed 61K. Default: 4096
ADTTIMEO	RECEIVE time out value Not used by the RCVFROM service Default: 0
ADTLPORT	Local well-known service port (used when ADTTOKEN=0) Indicates the local transport layer port that the calling application will be receiving (RCVFROM) datagrams on. If the SENDTO service creates the token, this port number is assigned by the transport layer and returned to the caller. if the RCVFROM service creates the token, this is the well-known port requested by the caller. If the RCVFROM service is creating the token, this value or the transport layer service name (ADTSRVCE) must be specified. This field is an unsigned positive integer with a maximum value of 65,534. The value must be unique for each server application. Default: None

PARAMETER	DESCRIPTION
ADTRPORT	Remote port Indicates the remote transport layer port on which the incoming datagram originated. This value is returned to the caller of the RCVFROM service and may be different for each datagram received. This field is an unsigned positive integer with a maximum value of 65,534. Default: None
ADTSRVCE	Transport layer service name (used when ADTTOKEN=0) Indicates the local transport layer service name. This string is mapped into a transport layer address or port, through internal DNR. The resolved value is the well-known port to which the client application sends (SENDTO) UDP datagrams. This field has a maximum length of 36 bytes. The value must be unique for each server application. This field is optional and is not modified by the RCVFROM service. Default: None
ADTSEP#	Number of separator characters for option ADTOPTN1_TYPSP. Not used in the RCVFROM service. Default: None
ADTSEP1	First or only spaceport character for option ADTOPTN1_TYPSP. Not used in the RCVFROM service. Default: None
ADTSEP2	Second character or separator sequence for option ADTOPTN1_TYPSP Not used in the RCVFROM service. Default: None
ADTLADDR	Local IP host address Indicates the local host internet address. This field is an unsigned four-byte integer value. The local host internet address is updated on establishment of a server connection, and will be returned to the caller. Default: None
ADTRADDR	Remote IP host address Indicates the remote host internet address. Either this field or the remote host name (ADTRNAME) field must be specified. This field is an unsigned four-byte integer value. The remote host internet address is updated when a server connection is established, and is returned to the caller. Default: None
ADTLNAME	Local IP host name Indicates the local host internet name. This field is a 255-byte character string that is padded with blanks. The local host internet name is updated when a client connection is established, and is returned to the caller. Default: None
ACMRNAME	Remote IP host name Indicates the remote host internet name. Either this value or the remote IP address (ADTRADDR) field must be specified. This field is a 255-byte character string that is padded with blanks. The remote host internet name is updated when a server connection is established, and is returned to the caller. Default: None
ADTUCNDX	One word of user context Specifies one arbitrary word of user context to be associated with the connection. The information provided is not interpreted by CPT, and is merely saved with other connection information. Default: 0 (no user context)

200801-024040-200100

PARAMETER	DESCRIPTION
ADTOPTN1	Specifies data transfer options These are the ADT options that apply to UDP data transfer requests: ◆ ADTOPTN1_DODNR– Execute internal DNR calls during UDP data transfer service routine calls (RCVFROM and SENDTO) to resolve remote IP addresses into IP names in the ADTRNAME field. ◆ ADTOPTN1_NBLKR – Do not block on a call to the RCVFROM service. If no datagrams are currently available, the return code, CPTWBLCK, will be returned in ADTRTNCD. Default: None **Note:** These options can be toggled on every UDP data transfer call even if the caller is using the same token.

Network Considerations

The ADT is a common data structure used for both client and server UDP applications. There are common and unique values specified for a particular service request.

The table describes network considerations for PL/1 API:

NAME	SERVER CONDITIONS FOR RCVFROM	CLIENT CONDITIONS FOR SENDTO
ADTLPORT	Local server well-known port selected by user application.	Local assigned transport provider port returned to user application.
ADTRPORT	Remote client transport provider port returned to user application.	Remote server transport provider well-known port selected by user application.
ADTSRVCE	Local server transport provider service name selected by user application.	Remote server transport provider service name selected by user application.
ADTRADDR	Remote IP host address returned to user application.	Remote IP host address selected by or returned to user application. The client must specify this field or ADTRNAME.
ADTLNAME	Local IP host name returned to user application.	Local IP host name Returned to user application.
ADTRNAME	Remote IP host name returned to user application only if ADTOPTN1_DODNR is specified in ADTOPTN1.	Remote IP host name selected by or returned to the user application. The client must specify this field or ADTRADDR. If ADTRADDR is used ADTRNAME will only be returned if ADTOPTN1_DODNR is specified in ADTOPTN1.

2C 024040-200100

Return Codes

The RECVFROM service return and diagnostic codes indicate the result of the execution. These values are in the ADTRTNCD and ADTDGNCD within the ADT. The diagnostic code is optional and generally indicates a transport provider return code. Read **Appendix C – MESSAGES AND CODES** in the ***CICS Programmer's Toolkit Installation and Administration Guide*** for the return code cross reference table.

This table describes the RECVFROM service return codes:

RETURN	DIAGNOSTIC CODE	DESCRIPTION
CPTIRCOK		Successful.
CPTEVERS		Control block version number is not supported.
CPTETOKN		Specified token is not valid.
CPTEBUFF		Buffer address and/or length is invalid.
CPTEPRGE	Yes	CPT Interface is terminating.
CPTENAPI	Yes	Transport provider API is not available.
CPTETERM	Yes	Environment is being terminated.
CPTERLSE	Yes	Release indication.
CPTEDISC	Yes	Disconnect indication.
CPTEENVR	Yes	Transport provider API environment error.
CPTEFRMT	Yes	Transport provider API format error.
CPTEPROC	Yes	Transport provider API procedure error.
CPTABEND		Abnormal exception occurred.
CPTEOTHR	Yes	An undefined exception occurred.

200801-024040-200100

Example:

In this example the application receives data from a remote host. The return code is checked to determine RECVFROM service completion status.

```
SAMP1: PROCEDURE OPTIONS (MAIN);

DCL    (T09FRCF)
       ENTRY OPTIONS (INTER ASSEMBLER);

DCL 1  CPT_ADT,
       %INCLUDE SYSLIB(T09KPADT);
       %INCLUDE SYSLIB(T09KPRCS);

DCL DATA BIT (1) INTI ('1'B);

DCL    MESSAGE CHAR (80);

%INCLUDE T09KPCON;

   /*
      Identify Service
    */

      ADTLPORT = 1580;

   DO WHILE (DATA);
   /*
      CPT RECEIVE Facility Management service request
    */
      ADTBUFFA = ADDR(MESSAGE);
      ADTBUFFL = ATG (MESSAGE);
      CALL T09FRCF (CPT_ADT);
      IF ADTRTNCD ^= 0
      THEN DO;/*
         Process and log RECEIVE error
         Terminate WHILE condition
       */
      END;

   /*
      Application to process data
    */

   END;
   /*
      CPT Terminate Endpoint service request
    */

   /*
      Terminate transaction
    */

      EXEC CICS RETURN;
```

RECEIVE

The RECEIVE service receives data from a peer transport user connected to an endpoint. The RECEIVE service receives data as input on a connection-mode (TCP) endpoint only.

To invoke the RECEIVE service, a user application must first build an Argument for Data Transfer (ADT) and then to issue a call for the RECEIVE routine. The ADT contains the version number, connection token, user buffer address, and length. When the RECEIVE service completes, the buffer length field is updated to reflect the amount of data processed by the RECEIVE service.

This table describes the RECEIVE service arguments:

INCLUDE STATEMENT	SIZE	CREATED BY
T09KPADT	644 (X'284')	User application

Structure

This is the structure of the RECEIVE service in PL/1 language:

```
DCL 1 CPT_ADT,
/*BEGIN %INCLUDE SYSLIB
(T09KPADT)**********************************
/*************************************************************/
/*                          CPT_ADT                          */
/*          CICS PROGRAMMER'S TOOLKIT DATA TRANSFER BLOCK */
/*-----------------------------------------------------------*/
5  ADTVERS    FIXED BIN (15) INIT(2),    /* ADT BLOCK VERSION  */
5  ADTFUNC    FIXED BIN (15) INIT(0),    /* FUNCTION CODE      */
5  ADTTOKEN   POINTER  INIT(NULL),       /* DATA XFER TOKEN    */
5  ADTBUFFA   POINTER  INIT(NULL),       /* DATA BUFFER ADDRSS */
5  ADTBUFFL   FIXED BIN (31) INIT(0),    /* BUFFER/DATA LENGTH */
5  ADTRTNCD   FIXED BIN (31) INIT(0),    /* RETURN CODE        */
5  ADTDGNCD   FIXED BIN (31) INIT(0),    /* DIAGNOSTIC CODE    */
5  ADTSFILL   CHAR  (3) INIT(LOW(3)),    /* RESERVED STATS     */
5  ADTSTATS   BIT (8) INIT('00000000'B), /* STATISTICS REQS    */
5  ADTTFILL   CHAR  (2) INIT(LOW(2)),    /* RESERVED TRACE     */
5  ADTTRAC2   BIT (8) INIT('00000000'B), /* TRACE OPTIONS 2    */
5  ADTTRAC1   BIT (8) INIT('00000000'B), /* TRACE OPTIONS 1    */
5  ADTQSEND   FIXED BIN (31) INIT(0),    /* TSEND QUEUE SIZE   */
5  ADTMSEND   FIXED BIN (31) INIT(0),    /* TSEND BUFFER SIZE  */
5  ADTQRECV   FIXED BIN (31) INIT(0),    /* TRECV QUEUE SIZE   */
5  ADTMRECV   FIXED BIN (31) INIT(0),    /* TRECV BUFFER SIZE  */
5  ADTTIMEO   FIXED BIN (31) INIT(0),    /* TIMED RECV SECONDS */
5  ADTRSVD1   FIXED BIN (31) INIT(0),    /*      RESERVED      */
5  ADTRSVD2   FIXED BIN (31) INIT(0),    /*      RESERVED      */
5  ADTRSVD3   FIXED BIN (31) INIT(0),    /*      RESERVED      */
5  ADTRSVD4   FIXED BIN (31) INIT(0),    /*      RESERVED      */
5  ADTLPORT   FIXED BIN (15) INIT(0),    /* LOCAL PORT         */
5  ADTRPORT   FIXED BIN (15) INIT(0),    /* REMOTE PORT        */
5  ADTRSVD5   FIXED BIN (15) INIT(0),    /*      RESERVED      */
5  ADTSRVCE       CHAR (36) INIT(' '),  /* SERVICE NAME       */
5  ADTSEP#    FIXED BIN (15) INIT(0),    /* # OF SEP CHARS     */
```

200801-024040-200100

```
5  ADTSEP1         CHAR  (1) INIT(' '),  /* 1ST/ONLY SEP CHAR  */
5  ADTSEP2         CHAR  (1) INIT(' '),  /* 2ND SEP CHARACTER  */
5  ADTRSVD6   FIXED BIN (15) INIT(0),     /*     RESERVED       */
5  ADTLADDR   FIXED BIN (31) INIT(0),     /* LOCAL HOST IP ADDR */
5  ADTRADDR   FIXED BIN (31) INIT(0),     /* REMOTE HOST IP ADR */
5  ADTLNAME       CHAR (255) INIT(' '),  /* LOCAL HOST NAME    */
5  ADTRSVD7        CHAR  (1) INIT(' '),  /*     RESERVED       */
5  ADTRNAME       CHAR (255) INIT(' '),  /* REMOTE HOST NAME   */
5  ADTRSVD8        CHAR  (1) INIT(' '),  /*     RESERVED       */
5  ADTUCNTX   FIXED BIN (31) INIT(0),    /* USER CONTEXT FIELD */
5  ADTOPTN4         BIT  (8) INIT('0'B), /*     RESERVED       */
5  ADTOPTN3         BIT  (8) INIT('0'B), /*     RESERVED       */
5  ADTOPTN2         BIT  (8) INIT('0'B), /* OPTIONS - BYTE 2   */
5  ADTOPTN1         BIT  (8) INIT('0'B); /* OPTIONS - BYTE 1   */
/*END    %INCLUDE SYSLIB
(T09KPADT)***********************************
```

The following constants are contained in the `T09MAC` library member, `T09KPCON`:

```
/*-------------------------------------------------------------
/*  ADTSTATS AND ADTTRACE VALUES ARE THE SAME AS DEFINED FOR   */
/*  THE ACM.  THESE FIELDS CAN THUS BE SET TO ACM-CONSTANTS.   */
/*-------------------------------------------------------------*/

         /*-------------------------------------------*/
            /*              ADT OPTIONS                */
         /*-------------------------------------------*/
ADTOPTN1_TYPSP BIT (8) INIT('00000001'B), /* RECD BY SEP CHAR*/
ADTOPTN1_TYPLL BIT (8) INIT('00000010'B), /* RECD BY LL PFX  */
ADTOPTN1_BLCKS BIT (8) INIT('00000100'B), /* BLOCKING SEND   */
ADTOPTN1_TMPRT BIT (8) INIT('00001000'B), /* TIMED PART RECV */
ADTOPTN1_TMRCV BIT (8) INIT('00010000'B), /* TIMED FULL RECV */
ADTOPTN1_NBLKR BIT (8) INIT('01000000'B), /* NO BLOCK RECV   */
ADTOPTN1_DODNR BIT (8) INIT('10000000'B), /* DO DNR FOR UDP  */
ADTOPTN2_NOSTP BIT (8) INIT('01000000'B), /* NO STRIP LL/SEP */
ADTOPTN2_VLIST BIT (8) INIT('10000000'B), /* VECTOR LIST     */
```

PARAMETER	DESCRIPTION
ADTVERS	Version Indicates the CPT version number of the argument used by the calling program. This required field is set to a binary 2 for this release of CPT. Default: None
ADTFUNC	Function code Indicates the function or callable service ID requested by the application program. This field is not set by the application, but is initialized by the **TRUE** interface stub program. Default: None

PARAMETER	DESCRIPTION
ADTTOKEN	Data transfer token Specifies a token that represents a TCP connection. If the ADT is being passed in a call to either the RECEIVE or SEND service, then it must be a token representing a previously established TCP connection, via the CONNECT or LISTEN service. it is an error to pass a zero ADTTOKEN to either the RECEIVE or SEND service. It is an error to pass a TCP token to the UDP data transfer service routines, RCVFROM and SENDTO. Conversely, it is an error to pass a UDP token to the TCP data transfer routines. RECEIVE and SEND. Default: 0
ADTBUFFA	User data address Indicates the storage address into which network data is placed. This is a contiguous segment of storage accessible to the user task. The storage area can be aligned on any boundary convenient for the application program. Default: 0
ADTBUFFL	User data length Indicates the length (in bytes) of user data in the storage area as identified by the ADTBUFFA operand. The length is updated when the request is completed to reflect the actual length of user data received. This field must be interpreted on completion to determine the amount of data actually received. If a RECEIVE request is issued with a zero length, an error is detected and the request fails. Default: 0
ADTRTNCD	Return code Indicates the return code set by the RECEIVE service. This value is returned and indicates the success or failure of the service. Default: 0
ADTDGNCD	Diagnostic code Indicates the diagnostic code set by the service request. This value generally indicates a transport provider return code. Default: 0
ADTSTATS	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
ADTTRACE	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
ADTQSEND	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
ADTMSEND	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
ADTQRECV	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
ADTMRECV	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.

200801-024040-200100

PARAMETER	DESCRIPTION
ADTTIMEO	RECEIVE timeout value Must be specified with these options: ◆ ADTOPTN1_TYPLL ◆ ADTOPTN1_TYPSP ◆ ADTOPTN1_TMRCV ◆ ADTOPTN1_TMPRT Specifying any of the above options on a RECEIVE call with an ADTTIMEO=ZERO will result in CPTETIME being returned in ADTRTNCD. Default: None
ADTLPORT	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
ADTRPORT	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
ADTSRVCE	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
ADTSEP#	Number of separator characters for option ADTOPTN1_TYPSP (0 < ADTSEP# < 3) If ADTSEP# is not equal to one or two, CPTESEP# will be returned in ADTRTNCD. Default: None
ADTSEP1	First or only separator character for option ADTOPTN1_TYPSP. Default: None
ADTSEP2	Second separator character in a sequence of two for option ADTOPTN1_TYPSP. Default: None
ADTLADDR	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
ADTRADDR	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
ADTLNAME	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
ADTUCNDX	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
ADTOPTN2	◆ ADTOPTN2_FVLST – Currently for internal use only. ◆ ADTOPTN2_NOSTP – Do not strip record delimiter sequence. ◆ This can be used with ADTTYPSP or ADTTYPLL to return the actual separator sequence or LL field in the buffer pointed to by ADTBUFFA

PARAMETER	DESCRIPTION
ADTOPTN1	This field specifies data transfer options. These are the ADT options that apply to TCP data transfer requests: ◆ ADTOPTN1_NBLKR – Do not block on a call to the RECEIVE service If no data is currently available on the connection, CPTWBLCK will be returned in ADTRTNCD. ◆ ADTOPTN1_TMRCV – Timed full record RECEIVE These fields (along with other required ADT fields) are used to request a timed full record RECEIVE: ADTOPTN1 = ADTOPTN1_TMRCV ADTTIMEO > zero ADTBUFFL set to the length expected If the time limit expires before receiving any or all of the data specified by ADTBUFFL, CPTWTIMO will be returned in ADTRTNCD along with any data that was received. ◆ ADTOPTN1_TMPRT – Timed partial record RECEIVE These fields (along with other required ADT fields) are used to request a timed partial record RECEIVE: ADTOPTN1 = ADTOPTN1_TMPRT ADTTIMEO > zero ADTBUFFL set to maximum length expected If the time limit expires before receiving data, CPTWTIMO is returned in ADTRTNCD. If the time limit expires and any data is received, the data, along with a zero ADTRTNCD, will be returned to the caller. ◆ ADTOPTN1_TYPSP – SEP type RECEIVE These fields (along with other required ADT fields) are used to request a SEP type RECEIVE call: ADTOPTN1 = ADTOPTN1_TYPSP ADTSEP# = 1 or 2 ADTSEP1 = character ADTSEP2 = character if ADTSEP# = 2 ADTTIMEO > zero If the time limit expires and data is received, but no SEP characters are found, the data, along with an ADTRTNCD of CPTWNSEP will be returned to the caller. ◆ ADTOPTN1_TYPLL – LL type RECEIVE These fields (along with other required ADT fields) are used to request a SEP type RECEIVE call: ADTOPTN1 = ADTOPTN1_TYPLL ADTTIMEO > zero If the time limit expires before receiving any or all of the data specified by the LL (first two bytes of the data stream), CPTWTIMO will be returned in ADTRTNCD, along with any data that was received. ◆ ADTOPTN1_BLCKS – Block on SEND service call (not used by RECEIVE service). **Note:** It is an error to combine any of these RECEIVE service options: ADTOPTN1_NBLKR, ADTOPTN1_TYPLL, ADTOPTN1_TYPSP, ADTOPTN1_TMRCV, ADTOPTN1_TMPRT. An invalid combination causes CPTEOPTN to return in ADTRTNCD. Default: None

Completion Information

The RECEIVE service completes normally when the data is moved from the transport provider buffer to the application program's storage area. A length is returned to the application program, which is set to the amount of data actually processed.

Normal completion of the RECEIVE service implies that data has been moved to the user buffer. This does not necessarily indicate the application request was completely satisfied, but that some amount of data was processed. The user application is required to load the ADTBUFFL field to determine the actual data received. The RECEIVE service returns control to the calling application on receipt of a full buffer, a partial buffer, or an error indication. Control is returned to the user application with a partial buffer to avoid a WAIT command within the RECEIVE service. Additional requests to the RECEIVE service may be required to completely satisfy the user application's requirement.

The presence of exceptions or error conditions do not always indicate serious errors. A user application should check the return code to determine proper flow control. The release indication return code is an example of a condition that is not necessarily a serious error. This exception specifies that the remote host has closed its half of the full-duplex data connection and will not send any additional data. This return code is acceptable, and generally indicates that graceful termination of the connection should begin.

On normal return to the application program, the general return code in ADTRTNCD is set to zero (CPTIRCOK). The diagnostic code in ADTDGNCD is always zero. The length field (ADTBUFFL) indicates the amount of data processed.

If the RECEIVE service completes abnormally, some or no user data may have been sent to the peer transport user. The general return code in ADTRTNCD, and the diagnostic code in ADTDGNCD indicate the nature of the failure. The diagnostic code (ADTDGNCD) generally contains a specific code that is generated by the transport provider.

20 024040-200100

Return Codes

The RECEIVE service return and diagnostic codes indicate the result of the execution. These values are in the ADTRTNCD and ADTDGNCD within the ADT. The diagnostic code is optional and generally indicates a transport provider return code. Read **Appendix C – MESSAGES AND CODES** in the ***CICS Programmer's Toolkit Installation and Administration Guide*** for the return code cross reference table.

This table describes the RECEIVE service return codes:

RETURN	DIAGNOSTIC CODE	DESCRIPTION
CPTIRCOK		Successful.
CPTEVERS		Control block version number is not supported.
CPTETOKN		Specified token is not valid.
CPTEBUFF		Buffer address and/or length is invalid.
CPTEPRGE	Yes	CPT Interface is terminating.
CPTENAPI	Yes	Transport provider API is not available.
CPTETERM	Yes	Environment is being terminated.
CPTERLSE	Yes	Release indication.
CPTEDISC	Yes	Disconnect indication.
CPTEENVR	Yes	Transport provider API environment error.
CPTEFRMT	Yes	Transport provider API format error.
CPTEPROC	Yes	Transport provider API procedure error.
CPTABEND		Abnormal exception occurred.
CPTEOTHR	Yes	An undefined exception occurred.

200801-024040-200100

Usage Information

The RECEIVE service receives normal data as input through a CPT connection. The data is part of a byte stream being received over a connection (TCP).

Data is moved from the transport provider's storage area to the user application's storage area. Stream data may not be received with the same logical boundaries with which it was sent. However, the data arrives in the precise order in which it was sent. Possible fragmentation is a characteristic of stream data.

User applications may be required to issue multiple RECEIVE service requests to obtain all of the desired data. The data may arrive in particle segments. An application should be designed to handle such a situation. Additionally, users who write applications to process multiple record oriented data should consider including a mechanism to delimit the data. Design options can include a logical length field at the beginning of a record, or a special field, or fields, at the end. This lets the application determine record boundaries.

The RECEIVE service request is a synchronous operation, which may require the application to be blocked. The RECEIVE queue (ACMQRECV) and buffer (ACMMRECV) sizes, as specified during connection initialization, control input queuing and buffering. The queue size represents the number of uncompleted RECEIVE requests that can be pending to the transport provider. The default value is generally sufficient to support most applications. The transport provider normally completes a RECEIVE request on transferring data into the user buffer. However, data over the network may be fragmented and may require multiple requests to retrieve all of the data. The RECEIVE service issues a WAIT command if no data is available.

The queue and buffer size values are specified during connection initialization and can be modified by either the LISTEN or CONNECT services. An application that is dependent on these values should validate the requested values, compared with those values returned within the ACM. The values are modified if the transport provider site administrator has configured limits and the application request exceeds those values. If the requested values are modified, verify site definition statements for API transport services.

The version number (ADTVERS) indicates the CPT release level in which this user application program is written. This required field is set to a binary 2 and is validated by the RECEIVE service before it processes the request.

The function code (ADTFUNC) indicates the CPT callable service ID. The field is not initialized by a user application program and has little value to the application except for dump analysis. The function code identifies and maps an argument list with the error or trace log and dump analysis.

The token (ADTTOKEN) indicates the connection that is to receive data. This is a required field and is validated by the RECEIVE service before it processes the request.

The data buffer address field, ADTBUFFA, is a full word. The application program assures that the residency mode of data areas it manages (for example, argument lists) is compatible with the addressing mode. The transport provider performs consistency checks on the addressing mode

20 024040-200100

whenever a service request is issued. However, unpredictable results can occur before the transport provider performs this check.

The data buffer length is indicated by the ADTBUFFL field. This is a full word positive integer. The data buffer length field should be less than or equal to the maximum receive buffer values. However, if the data buffer length is greater than the maximum receive buffer, the RECEIVE service attempts to satisfy the user's request with multiple transport provider requests. On return from the RECEIVE service, the ADTBUFFL is updated with a value that indicates the number of bytes processed.

The ADTOPTN1 field specifies RECEIVE processing control options and provides a mechanism for event notification on return to the application program.

200801-024040-200100

Example:

In this example a has been established connection and the application receives data from the remote host. The token is loaded from the ACM and used by all of the following CPT service requests. The return code is checked to determine RECEIVE service completion status.

```
SAMP1: PROCEDURE OPTIONS (MAIN);

DCL    (T09FREC)
       ENTRY OPTIONS (INTER ASSEMBLER);
DCL    (CPTPRT,MSGPRT) POINTER;

DCL 1  CPT_ADT,
       %INCLUDE SYSLIB(T09KPADT);
DCL 1  CPT_RCS,
       %INCLUDE SYSLIB(T09KPRCS);

DCL DATA BIT (1) INTI ('1'B);

DCL    MESSAGE CHAR (80);

%INCLUDE T09KPCON;

   /*
      CPT Connection Management service request
    */

      ADTTOKEN = ACMTOKEN;

   DO WHILE (DATA);
   /*
      CPT RECEVIE Facility Management service request
    */
      ADTBUFFA = ADDR(BUFFER);
      ADTBUFFL = 80;
      CALL T09FREC (CPT_ADT);
      IF ADTRTNCD ^= 0
      THEN DO;/*
         Process and log RECEIVE error
         Terminate WHILE condition
       */
      END;

   /*
      Application to process data
    */

   END;
   /*
      CPT Connection Release service request
    */

   /*
      Terminate transaction
    */

      EXECCICS RETURN;
```

SEND

The SEND service sends data to a peer transport user connected to an endpoint. The SEND service sends data as output on a connection-mode (TCP) endpoint only.

To invoke the SEND service, a user application is required to first build an ADT and then to issue a call for the SEND routine. The ADT contains the version number, connection token, user buffer address, and length. When the SEND service completes, the buffer length field is updated to reflect the amount of data processed.

This table describes the SEND service arguments:

INCLUDE STATEMENT	SIZE	CREATED BY
T09KPADT	644 (X'284')	User application

Structure

This is the structure of the SEND service in PL/1 language:

```
DCL 1 CPT_ADT,
/*BEGIN %INCLUDE SYSLIB
(T09KPADT)**********************************
/*************************************************************/
/*                          CPT_ADT                          */
/*         CICS PROGRAMMER'S TOOLKIT DATA TRANSFER BLOCK */
/*-----------------------------------------------------------*/
5  ADTVERS    FIXED BIN (15) INIT(2),    /* ADT BLOCK VERSION  */
5  ADTFUNC    FIXED BIN (15) INIT(0),    /* FUNCTION CODE      */
5  ADTTOKEN   POINTER  INIT(NULL),       /* DATA XFER TOKEN    */
5  ADTBUFFA   POINTER  INIT(NULL),       /* DATA BUFFER ADDRSS */
5  ADTBUFFL   FIXED BIN (31) INIT(0),    /* BUFFER/DATA LENGTH */
5  ADTRTNCD   FIXED BIN (31) INIT(0),    /* RETURN CODE        */
5  ADTDGNCD   FIXED BIN (31) INIT(0),    /* DIAGNOSTIC CODE    */
5  ADTSFILL   CHAR  (3) INIT(LOW(3)),    /* RESERVED STATS     */
5  ADTSTATS   BIT (8) INIT('00000000'B), /* STATISTICS REQS    */
5  ADTTFILL   CHAR  (2) INIT(LOW(2)),    /* RESERVED TRACE     */
5  ADTTRAC2   BIT (8) INIT('00000000'B), /* TRACE OPTIONS 2    */
5  ADTTRAC1   BIT (8) INIT('00000000'B), /* TRACE OPTIONS 1    */
5  ADTQSEND   FIXED BIN (31) INIT(0),    /* TSEND QUEUE SIZE   */
5  ADTMSEND   FIXED BIN (31) INIT(0),    /* TSEND BUFFER SIZE  */
5  ADTQRECV   FIXED BIN (31) INIT(0),    /* TRECV QUEUE SIZE   */
5  ADTMRECV   FIXED BIN (31) INIT(0),    /* TRECV BUFFER SIZE  */
5  ADTTIMEO   FIXED BIN (31) INIT(0),    /* TIMED RECV SECONDS */
5  ADTRSVD1   FIXED BIN (31) INIT(0),    /*      RESERVED      */
5  ADTRSVD2   FIXED BIN (31) INIT(0),    /*      RESERVED      */
5  ADTRSVD3   FIXED BIN (31) INIT(0),    /*      RESERVED      */
5  ADTRSVD4   FIXED BIN (31) INIT(0),    /*      RESERVED      */
5  ADTLPORT   FIXED BIN (15) INIT(0),    /* LOCAL PORT         */
5  ADTRPORT   FIXED BIN (15) INIT(0),    /* REMOTE PORT        */
5  ADTRSVD5   FIXED BIN (15) INIT(0),    /*      RESERVED      */
5  ADTSRVCE        CHAR (36) INIT(' '),  /* SERVICE NAME       */
5  ADTSEP#    FIXED BIN (15) INIT(0),    /* # OF SEP CHARS     */
```

200801-024040-200100

```
5  ADTSEP1         CHAR  (1) INIT(' '),  /* 1ST/ONLY SEP CHAR  */
5  ADTSEP2         CHAR  (1) INIT(' '),  /* 2ND SEP CHARACTER  */
5  ADTRSVD6   FIXED BIN (15) INIT(0),    /*     RESERVED       */
5  ADTLADDR   FIXED BIN (31) INIT(0),    /* LOCAL HOST IP ADDR */
5  ADTRADDR   FIXED BIN (31) INIT(0),    /* REMOTE HOST IP ADR */
5  ADTLNAME       CHAR (255) INIT(' '),  /* LOCAL HOST NAME    */
5  ADTRSVD7        CHAR  (1) INIT(' '),  /*     RESERVED       */
5  ADTRNAME       CHAR (255) INIT(' '),  /* REMOTE HOST NAME   */
5  ADTRSVD8        CHAR  (1) INIT(' '),  /*     RESERVED       */
5  ADTUCNTX   FIXED BIN (31) INIT(0),    /* USER CONTEXT FIELD */
5  ADTOPTN4         BIT  (8) INIT('0'B), /*     RESERVED       */
5  ADTOPTN3         BIT  (8) INIT('0'B), /*     RESERVED       */
5  ADTOPTN2         BIT  (8) INIT('0'B), /* OPTIONS - BYTE 2   */
5  ADTOPTN1         BIT  (8) INIT('0'B); /* OPTIONS - BYTE 1   */
/*END   %INCLUDE SYSLIB
(T09KPADT)**********************************
```

The following constants are contained in the `T09MAC` library member, `T09KPCON`:

```
/*-------------------------------------------------------------
/*  ADTSTATS AND ADTTRACE VALUES ARE THE SAME AS DEFINED FOR   */
/*  THE ACM.  THESE FIELDS CAN THUS BE SET TO ACM-CONSTANTS.   */
/*-------------------------------------------------------------*/

        /*---------------------------------------------*/
           /*              ADT OPTIONS                */
        /*---------------------------------------------*/
ADTOPTN1_TYPSP BIT (8) INIT('00000001'B), /* RECD BY SEP CHAR*/
ADTOPTN1_TYPLL BIT (8) INIT('00000010'B), /* RECD BY LL PFX  */
ADTOPTN1_BLCKS BIT (8) INIT('00000100'B), /* BLOCKING SEND   */
ADTOPTN1_TMPRT BIT (8) INIT('00001000'B), /* TIMED PART RECV */
ADTOPTN1_TMRCV BIT (8) INIT('00010000'B), /* TIMED FULL RECV */
ADTOPTN1_NBLKR BIT (8) INIT('01000000'B), /* NO BLOCK RECV   */
ADTOPTN1_DODNR BIT (8) INIT('10000000'B), /* DO DNR FOR UDP  */
ADTOPTN2_NOSTP BIT (8) INIT('01000000'B), /* NO STRIP LL/SEP */
ADTOPTN2_VLIST BIT (8) INIT('10000000'B), /* VECTOR LIST     */
```

PARAMETER	DESCRIPTION
ADTVERS	Version Indicates the CPT version number of the argument used by the calling program. This required field is set to a binary 2 for this release of CPT. Default: None
ADTFUNC	Function code Indicates the function or callable service ID requested by the application program. This field is not set by the application, but is initialized by the **TRUE** interface stub program. Default: None

2024040-200100

PARAMETER	DESCRIPTION
ADTTOKEN	Data transfer token Specifies a token that represents a TCP connection. If the ADT is being passed in a call to either the RECEIVE or SEND service, then it must be a token representing a previously established TCP connection, via the CONNECT or LISTEN service. it is an error to pass a zero ADTTOKEN to either the RECEIVE or SEND service. It is an error to pass a TCP token to the UDP data transfer service routines, RCVFROM and SENDTO. Conversely, it is an error to pass a UDP token to the TCP data transfer routines. RECEIVE and SEND. Default: 0
ADTBUFFA	User data address Indicates the storage address into which network data is placed. This is a contiguous segment of storage accessible to the user task. The storage area can be aligned on any boundary convenient for the application program. Default: 0
ADTBUFFL	User data length Indicates the length (in bytes) of user data in the storage area as identified by the ADTBUFFA operand. The length is updated when the request is completed to reflect the actual length of user data received. This field must be interpreted on completion to determine the amount of data actually received. If a SEND request is issued with a zero length, an error is detected and the request fails. Default: 0
ADTRTNCD	Return code Indicates the return code set by the SEND service. This value is returned and indicates the success or failure of the service. Default: 0
ADTDGNCD	Diagnostic code Indicates the diagnostic code set by the service request. This value generally indicates a transport provider return code. Default: 0
ADTSTATS	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
ADTTRACE	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
ADTQSEND	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
ADTMSEND	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
ADTQRECV	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
ADTMRECV	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
ADTTIMEO	RECEIVE timeout value Not used by the SEND service. Default: None
ADTLPORT	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
ADTRPORT	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.

200801-024040-200100

PARAMETER	DESCRIPTION
ADTSRVCE	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
ADTSEP#	Number of separator characters for option ADTOPTN1_TYPSP (0 < ADTSEP# < 3) If ADTSEP# is not equal to one or two, CPTESEP# will be returned in ADTRTNCD. Default: None
ADTSEP1	First or only separator character for option ADTOPTN1_TYPSP. Default: None
ADTSEP2	Second separator character in a sequence of two for option ADTOPTN1_TYPSP. Default: None
ADTLADDR	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
ADTRADDR	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
ADTLNAME	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
ADTUCNDX	This field is used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.

ADTOPTN1	This field specifies data transfer options.
	These are the ADT options that apply to TCP data transfer requests:
	◆ ADTOPTN1_NBLKR – Do not block on a call to the RECEIVE service. Not used by the SEND service.
	◆ ADTOPTN1_TMRCV – Timed full record RECEIVE. Not used by the SEND service.
	◆ ADTOPTN1_TMPRT – Timed partial record RECEIVE. Not used by the SEND service.
	◆ ADTOPTN1_TYPSP – SEP type SEND. These files (along with other required ADT fields) are used to request a SEP type SEND call:
	`adt_optn1 = ADTOPTN1_TYPSP`
	`adt_sepc = 1 OR 2`
	`adt_sep1 = character`
	`adt_sep2 = character if ADTSEP# = 2`
	◆ ADTOPTN1_TYPLL – LL type SEND. These fields (along with other required ADT fields) are used to request a SEP type SEND call:
	`adt_optn1 = ADTOPTN1_TYPLL`
	◆ ADTOPTN1_BLCKS – Block on SEND service call. The default for the SEND service is to return control to the caller as soon as the SEND request has been scheduled with the local transport provider. Option ADTOPTN1_BLCKS will block the return to the caller until the SEND data has been moved to the local transport provider's address space (not the remote TCP). This can provide TCP connection status at the SEND service call time rather than at some later time (next SEND or RECEIVE call).
	Note: It is an error to combine any of these SEND service options:
	ADTOPTN1_TYPLL
	ADTOPTN1_TYPSP
	An invalid combination will result in CPTEOPTN being returned in ADTRTNCD.
	Default: None

Completion Information

The SEND service completes normally when the data has been both moved from the application program's storage area and forwarded to the transport provider for sending to the connected transport user. A length is returned to the application program, which is set to the amount of data processed.

Normal completion of the SEND service implies nothing in regard to when the data is sent to the peer transport user. This only means that the transport provider has taken custody of the user data and the storage area provided by the application program can be reused. The transport provider generally sends the buffered data, but this may not occur synchronously with the completion of the SEND service.

On normal return to the application program, the general return code in ADTRTNCD is set to zero (CPTIRCOK). The diagnostic code in ADTDGNCD is always zero. The length field (ADTBUFFL) indicates the amount of data processed.

If the SEND service completes abnormally, some or no user data may have been sent to the peer transport user. The general return code in ADTRTNCD and the diagnostic code in ADTDGNCD indicate the nature of the failure. The diagnostic code (ADTDGNCD) generally contains a specific code that is generated by the transport provider.

200801-024040-200100

Return Codes

The SEND service return and diagnostic codes indicate the result of the execution. These values are in the ADTRTNCD and ADTDGNCD within the ADT. The diagnostic code is optional and generally indicates a transport provider return code. Read **Appendix C – MESSAGES AND CODES** in the ***CICS Programmer's Toolkit Installation and Administration Guide*** for the return code cross reference table.

This table describes the SEND service return codes:

RETURN	DIAGNOSTIC CODE	DESCRIPTION
CPTIRCOK		Successful.
CPTEVERS		Control block version number is not supported.
CPTETOKN		Specified token is not valid.
CPTEBUFF		Buffer address and/or length is invalid.
CPTEPRGE	Yes	CPT Interface is terminating.
CPTENAPI	Yes	Transport provider API is not available.
CPTETERM	Yes	Environment is being terminated.
CPTERLSE	Yes	Release indication.
CPTEDISC	Yes	Disconnect indication.
CPTEENVR	Yes	Transport provider API environment error.
CPTEFRMT	Yes	Transport provider API format error.
CPTEPROC	Yes	Transport provider API procedure error.
CPTABEND		Abnormal exception occurred.
CPTEOTHR	Yes	An undefined exception occurred.

Usage Information

The SEND service sends normal data as output through a CPT connection. The data is part of a byte stream being sent over a connection (TCP).

Data is moved from the application program's storage area to storage areas maintained by the transport provider. The data is packetized and sent to the connection transport user. Logical boundaries are not preserved in the data stream. The data is delivered to the peer transport user in the precise order in which it was sent. However, this data may be fragmented.

Data is not necessarily packetized and sent by the transport provider each time a SEND service is issued, nor is it sent when a buffer boundary is indicated. The transport provider may intentionally delay sending data as the result of performance optimization or congestion avoidance algorithms. Normally, data generated by the application is forwarded when it is sent in a continuous flow.

The SEND service request is a synchronous operation that may require the application to be blocked. The SEND queue (ACMQSEND) and buffer (ACMMSEND) sizes, as specified during connection initialization, control output queuing and buffering. The queue size represents the number of uncompleted SEND requests that can be pending to the transport provider. The default value is generally sufficient to support most applications. The transport provider normally completes a SEND request within a reasonable amount of time, but a congested network may cause the transport provider delays in completing requests, and cause the application to be blocked.

Additionally, for an application that issues multiple SEND requests contiguously, consider increasing the default queue size. The buffer size represents the maximum number of user data bytes that can be transferred by the application in a single SEND request to the transport provider. This value is application dependent. A small value causes the SEND service to issue multiple transport provider SEND requests. Multiple transport provider SEND requests do not present a problem, but do reduce the queue size and can cause the application to be blocked. A large buffer value can waste application storage.

The queue and buffer size values are specified during connection initialization and can be modified on return. An application that is dependent on these values should validate the requested values compared with values returned within the ACM. The values are modified if the transport provider site administrator has configured limits and the application request exceeds those values. If the requested values are modified, verify site definition statements for API transport services.

The version number (ADTVERS) indicates the CPT release level in which this user application program is written. This required field is set to a binary 2 and is validated by the SEND service before processing the request.

The function code (ADTFUNC) indicates the CPT callable service ID. This field is not initialized by a user application program and has little value to the application except for dump analysis. The function code can identify and map an argument list with the error or trace log and dump analysis.

The token (ADTTOKEN) indicates the connection that will transmit data. This required field is validated by the SEND service before it processes the request.

200801-024040-200100

The data buffer address field ADTBUFFA is a full word. The application program assures that the residency mode of data areas it manages (for example, argument lists) is compatible with the addressing mode. The transport provider performs consistency checks on the addressing mode whenever a service request is issued. However, unpredictable results may occur before the transport provider performs this check.

The data buffer length is indicated by the ADTBUFFL field. This is a full word positive integer. The data buffer length field should be less than or equal to the maximum send buffer values. However, if the data buffer length is greater than the maximum send buffer, the SEND service will fragment the user data into multiple transport provider requests. The ADTBUFFL is updated on return from the SEND service with a value that indicates the number of bytes processed.

The ADTOPTN1 field specifies SEND processing control options and provides a mechanism for event notification on return to the application program.

2024040-200100

Example:

This example establishes a connection and sends a welcome message to the remote host. The token is loaded from the ACM and used by all of the following CPT service requests. The return code is checked to determine SEND service completion status.

```
SAMP1: PROCEDURE OPTIONS (MAIN);

DCL    (T09FSEN)
       ENTRY OPTIONS (INTER ASSEMBLER);
DCL    (CPTPRT,MSGPRT) POINTER;

DCL 1  CPT_ADT,
       %INCLUDE SYSLIB(T09KPADT);
DCL 1  CPT_RCS,
       %INCLUDE SYSLIB(T09KPRCS);

DCL DATA BIT (1) INTI ('1'B);

DCL    MESSAGE CHAR (80)
   INIT('WELCOME TO THE CICS/API PROGRAMMING WORLD');;

%INCLUDE T09KPCON;

   /*
      CPT Connection Management service request
    */

      ADTTOKEN = ACMTOKEN;

   DO WHILE (DATA);

      /*
         Application to process data
       */
      /*
         CPT SEND Facility Management service request
       */
            ADTBUFFA = ADDR(BUFFER);
            ADTBUFFL = 80;
            CALL T09FSEN (CPT_ADT);
            IF ADTRTNCD ^= 0
         THEN DO;
         /*
            Process and log SEND error
            Terminate WHILE condition
          */
         END;


   END;
   /*
      CPT Connection Release service request
    */

   /*
      Terminate transaction
    */
      EXEC   CICS RETURN;
```

SENDTO

This service is provided to allow connectionless client and server applications to be developed. This service is UDP only. The SENDTO service provides two basic functions:

- Establish a UDP client endpoint represented by a new token and send a datagram to a remote UDP server. This function is indicated to the SENDTO service by passing an ADTTOKEN equal to zero. SENDTO will then create all the internal control blocks and the SENDTO buffer queue. Even though the RCVFROM buffer queue will not be allocated for this endpoint (token) until the RCVFROM service is called, the RCVFROM buffer size and number must be specified at this time because they are negotiated with the transport provider and recorded in the internal CPT control blocks at endpoint creation time. Upon return from the SENDTO service ADTTOKEN will contain the token value to be passed to subsequent SENDTO and RCVFROM service calls.
- Send a datagram at a previously established UDP endpoint represented by an existing token. This functionality makes the SENDTO service call just a data transfer call that can be used by a client or server application. The SENDTO buffer queue is only allocated upon the first call to the SENDTO service whether ADTTOKEN is equal to zero or not.

UDP tokens created with the RCVFROM or SENDTO services cannot be passed to the TCP only services, CONNECT, LISTEN, SEND, and RECEIVE. All other CPT service calls are available to UDP applications.

This table describes the SENDTO service arguments:

INCLUDE STATEMENT	SIZE	CREATED BY
T09KPADT	644 (X'284')	User application

20-024040-200100

Structure

This is the structure of the SENDTO service in PL/1 language:

```
DCL 1 CPT_ADT,
/*BEGIN %INCLUDE SYSLIB
(T09KPADT)**********************************
/*************************************************************/
/*                          CPT_ADT                          */
/*         CICS PROGRAMMER'S TOOLKIT DATA TRANSFER BLOCK */
/*-----------------------------------------------------------*/
5  ADTVERS    FIXED BIN (15) INIT(2),    /* ADT BLOCK VERSION  */
5  ADTFUNC    FIXED BIN (15) INIT(0),    /* FUNCTION CODE      */
5  ADTTOKEN   POINTER  INIT(NULL),       /* DATA XFER TOKEN    */
5  ADTBUFFA   POINTER  INIT(NULL),       /* DATA BUFFER ADDRSS */
5  ADTBUFFL   FIXED BIN (31) INIT(0),    /* BUFFER/DATA LENGTH */
5  ADTRTNCD   FIXED BIN (31) INIT(0),    /* RETURN CODE        */
5  ADTDGNCD   FIXED BIN (31) INIT(0),    /* DIAGNOSTIC CODE    */
5  ADTSFILL   CHAR  (3) INIT(LOW(3)),    /* RESERVED STATS     */
5  ADTSTATS   BIT (8) INIT('00000000'B), /* STATISTICS REQS    */
5  ADTTFILL   CHAR  (2) INIT(LOW(2)),    /* RESERVED TRACE     */
5  ADTTRAC2   BIT (8) INIT('00000000'B), /* TRACE OPTIONS 2    */
5  ADTTRAC1   BIT (8) INIT('00000000'B), /* TRACE OPTIONS 1    */
5  ADTQSEND   FIXED BIN (31) INIT(0),    /* TSEND QUEUE SIZE   */
5  ADTMSEND   FIXED BIN (31) INIT(0),    /* TSEND BUFFER SIZE  */
5  ADTQRECV   FIXED BIN (31) INIT(0),    /* TRECV QUEUE SIZE   */
5  ADTMRECV   FIXED BIN (31) INIT(0),    /* TRECV BUFFER SIZE  */
5  ADTTIMEO   FIXED BIN (31) INIT(0),    /* TIMED RECV SECONDS */
5  ADTRSVD1   FIXED BIN (31) INIT(0),    /*      RESERVED      */
5  ADTRSVD2   FIXED BIN (31) INIT(0),    /*      RESERVED      */
5  ADTRSVD3   FIXED BIN (31) INIT(0),    /*      RESERVED      */
5  ADTRSVD4   FIXED BIN (31) INIT(0),    /*      RESERVED      */
5  ADTLPORT   FIXED BIN (15) INIT(0),    /* LOCAL PORT         */
5  ADTRPORT   FIXED BIN (15) INIT(0),    /* REMOTE PORT        */
5  ADTRSVD5   FIXED BIN (15) INIT(0),    /*      RESERVED      */
5  ADTSRVCE       CHAR (36) INIT(' '),  /* SERVICE NAME       */
5  ADTSEP#    FIXED BIN (15) INIT(0),    /* # OF SEP CHARS     */
5  ADTSEP1        CHAR  (1) INIT(' '),  /* 1ST/ONLY SEP CHAR  */
5  ADTSEP2        CHAR  (1) INIT(' '),  /* 2ND SEP CHARACTER  */
5  ADTRSVD6   FIXED BIN (15) INIT(0),    /*     RESERVED      */
5  ADTLADDR   FIXED BIN (31) INIT(0),    /* LOCAL HOST IP ADDR */
5  ADTRADDR   FIXED BIN (31) INIT(0),    /* REMOTE HOST IP ADR */
5  ADTLNAME      CHAR (255) INIT(' '),  /* LOCAL HOST NAME    */
5  ADTRSVD7       CHAR  (1) INIT(' '),  /*     RESERVED       */
5  ADTRNAME      CHAR (255) INIT(' '),  /* REMOTE HOST NAME   */
5  ADTRSVD8       CHAR  (1) INIT(' '),  /*     RESERVED       */
5  ADTUCNTX   FIXED BIN (31) INIT(0),    /* USER CONTEXT FIELD */
5  ADTOPTN4        BIT  (8) INIT('0'B), /*     RESERVED       */
5  ADTOPTN3        BIT  (8) INIT('0'B), /*     RESERVED       */
5  ADTOPTN2        BIT  (8) INIT('0'B), /* OPTIONS - BYTE 2   */
5  ADTOPTN1        BIT  (8) INIT('0'B); /* OPTIONS - BYTE 1   */
/*END   %INCLUDE SYSLIB
(T09KPADT)**********************************
```

The following constants are contained in the T09MAC library member, T09KPCON:

200801-024040-200100

```
/*-------------------------------------------------------------
/*  ADTSTATS AND ADTTRACE VALUES ARE THE SAME AS DEFINED FOR   */
/*  THE ACM.  THESE FIELDS CAN THUS BE SET TO ACM-CONSTANTS.   */
/*-------------------------------------------------------------*/

        /*-------------------------------------------*/
           /*              ADT OPTIONS               */
        /*-------------------------------------------*/
ADTOPTN1_TYPSP BIT (8) INIT('00000001'B), /* RECD BY SEP CHAR*/
ADTOPTN1_TYPLL BIT (8) INIT('00000010'B), /* RECD BY LL PFX  */
ADTOPTN1_BLCKS BIT (8) INIT('00000100'B), /* BLOCKING SEND   */
ADTOPTN1_TMPRT BIT (8) INIT('00001000'B), /* TIMED PART RECV */
ADTOPTN1_TMRCV BIT (8) INIT('00010000'B), /* TIMED FULL RECV */
ADTOPTN1_NBLKR BIT (8) INIT('01000000'B), /* NO BLOCK RECV   */
ADTOPTN1_DODNR BIT (8) INIT('10000000'B), /* DO DNR FOR UDP  */
ADTOPTN2_NOSTP BIT (8) INIT('01000000'B), /* NO STRIP LL/SEP */
ADTOPTN2_VLIST BIT (8) INIT('10000000'B), /* VECTOR LIST     */
```

PARAMETER	DESCRIPTION
ADTVERS	Version Indicates the CPT version number of the argument used by the calling program. This required field is set to a binary 2 for this release of CPT. Default: None
ADTFUNC	Function code Indicates the function or callable service ID requested by the application program. This field is not set by the application, but is initialized by the TRUE interface stub program. Default: None
ADTTOKEN	Data transfer token Specifies a token that represents a UDP endpoint. If the ADT is being passed in a call to either the RCVFROM or SENDTO service, the token can be zero, indicating to either service, to first create a token before sending or receiving a datagram. If the token is not zero, it must be a token created previously by either the RCVFROM or SENDTO service. It is not necessary or efficient to create a token every time a CICS transaction calls the UDP data transfer services. It is an error to pass a TCP token to the UDP data transfer service routines, RCVFROM and SENDTO. Conversely, it is an error to pass a UDP token to the TCP data transfer routines, RECEIVE and SEND. Default: 0
ADTBUFFA	User data address Indicates the storage address from which the UDP datagram will be sent (SENDTO service). This is a contiguous segment of storage accessible to the user task. The content of all user data is application dependent, and not interpreted by either CPT nor the transport provider. The storage area can be aligned on any boundary convenient for the application program. Default: 0
ADTBUFFL	User data length Indicates the length (in bytes) of the buffer specified in ADTBUFFA which is to be sent (SENDTO service). It is an error to call the SENDTO service with an ADTBUFFL of zero. Upon return to the caller, ADTBUFFL reflects the number of bytes actually sent (generally the number requested). Indicates the return code set by the SENDTO service. Default: 0
ADTRTNCD	Return code Indicates the return code set by the SENDTO service. This value is also returned in register 15 and indicates the success or failure of the service. Default: 0

PARAMETER	DESCRIPTION
ADTDGNCD	Diagnostic code Indicates the diagnostic code set by the service request. This value generally indicates a transport provider return code. Default: 0
ADTSTATS	ADTSTATS_CONN | ADTSTATS_TERM Specifies statistics logging options for the application program. This facility can be used for debugging and tuning during development. ◆ ADTSTATS_CONN – Specifies that a message(s) be generated when either a listen service or a data transfer connection is established. These messages are generated by the CPT LISTEN and CONNECT services. The message numbers associated with this option are CPT802I and CPT803I. ◆ ADTSTATS_TERM – Specifies that a message(s) be generated on terminating an established connection. These messages are generated by the CPT CLOSE service. The message numbers associated with this option are CPT807I, CPT808I, and CPT809I. Default: 0 (no statistics logging)
ADTTRAC1	Specifies trace logging options for the application program. This facility can be used for debugging during development. ◆ ADTTRAC1_NTRY – Specifies that a message be generated on entry to a CPT service routine. Messages are generated by all CPT service routines. The message number associated with this option is CPT901I. ◆ ADTTRAC1_ARGS – Specifies that a hex dump of the caller's arguments be generated on entry and exit of a CPT service routine. Messages are generated by the corresponding service routine. The message numbers associated with this option are CPT902I, CPT903I, CPT904I, CPT911I, CPT912, CPT921I, CPT922I, CPT924I, CPT925I, CPT930I, and CPT932I. ◆ ADTTRAC1_RECV – Specifies that a hex dump of the transport provider's input (RECEIVE) data be logged. Messages are generated by the CPT RECV service. The message number associated with this option is CPT913I. ◆ ADTTRAC1_SEND – Specifies that a hex dump of the transport provider's output (SEND) data be logged. Messages are generated by the CPT SEND service. The message number associated with this option is CPT914I. ◆ ADTTRAC1_TERM – Specifies that a message be generated on termination of a CPT service routine. Messages are generated by all CPT service routines. The message number associated with this option is CPT908I. ◆ ADTTRAC1_PASS – Specifies that a hex dump of resources related to a passed connection be logged. Messages are generated by the CPT GIVE and TAKE services. The message numbers associated with this option are CPT919I and CPT920I. ◆ ADTTRAC1_CLSE – Specifies that a hex dump of resources related to a CLOSE processing of a connection be logged. Messages are generated by the CPT CLOSE service. The message number associated with this option is CPT906I ◆ ADTTRAC1_TERR – Specifies that a hex dump of a transport provider API parameter list that fails successful completion be logged. The message number associated with this option is CPT400I.

200801-024040-200100

PARAMETER	DESCRIPTION
ADTTRAC2	◆ ADTTRAC2_TOKN – Specifies that a hex dump of the connection token be logged. Messages are generated by all services routines on entry. The message number associated with this option is CPT909I. ◆ ADTTRAC2_TPL – Specifies that a hex dump of the transport provider API parameter list be logged. The message numbers associated with this option are CPT917I and CTP935I. ◆ ADTTRAC2_RLSE – Specifies that a message indicating a transport provider release indication is to be logged. Messages are generated by various CPT service routines. The message number associated with this option is CPT414I. ◆ ADTTRAC2_STOR – Specifies that a hex dump of storage management argument to be logged. Messages are generated by various CPT service routines. The message numbers associated with this option are CPT928I and CPT929I. Default: 0 (no trace logging)
ADTQSEND	API send queue size (used when ADTTOKEN=0) Specifies the maximum number of uncompleted SENDTO requests that can be queued by the application to the transport provider (API). This value lets applications control output processing and can affect throughput rates. The value is negotiated with the transport provider and can be modified by the transport provider. The total allocation for output processing is the product of the SENDTO queue and buffer size values and cannot exceed 61K. Default: 4
ADTMSEND	API send buffer size (used when ADTTOKEN=0) Specifies the maximum number of user data bytes that can be transferred by the application in a single SENDTO request to the transport provider (API). This value lets applications control output processing and can affect throughput rates. The value is negotiated with the transport provider and can be modified by the transport provider. The total allocation for output processing is the product of the SENDTO queue and buffer size values and cannot exceed 61K. Default: 4096
ADTQRECV	API receive queue size (used when ADTTOKEN=0) Specifies maximum number of uncompleted RCVFROM requests that can be queued by the application to the transport provider (API). This value lets applications control input processing and can affect throughput rates. The value is negotiated with the transport provider and can be modified by the transport provider. The total allocation for input processing is the product of the RCVFROM queue and buffer size values and cannot exceed 61K. Default: 4
ADTMRECV	API RECEIVE buffer size (used when ADTTOKEN=0) Specifies the maximum number of user data bytes that can be transferred by the application in a single RCVFROM request to the transport provider (API). This value lets applications control input processing and can affect throughput rates. The value is negotiated with the transport provider and can be modified by the transport provider. The total allocation for input processing is the product of the RCVFROM queue and buffer size values and cannot exceed 61K. Default: 4096
ADTTIMEO	RECEIVE timeout value Not used by the SENDTO service Default: 0
ADTLPORT	Local well-known service port. Indicates the local transport layer port that the calling application will be sending (SENDTO) UDP datagrams from. If the SENDTO service creates the token, this port number is assigned by the transport layer and returned to the caller. If the RCVFROM service creates the token, this is the well-known port requested by the caller. This field is an unsigned positive integer with a maximum value of 65,534. The value must be unique for each server application. Default: None

PARAMETER	DESCRIPTION
ADTRPORT	Remote port. Indicates the remote transport layer port destination for the datagram being processed by the SENDTO service. This field is an unsigned positive integer with a maximum value of 65,534. Default: None
ADTSRVCE	Transport layer service name Not used in the SENDTO. Default: None
ADTSEP#	Number of separator characters for option ADTOPTNS_TYPSP. Not used in the SENDTO service. Default: None
ADTSEP1	First or only separator character for option ADTOPTNS_TYPSP Not used in the SENDTO service. Default: None
ADTSEP2	Second character or separator sequence for option ADTOPTNS_TYPSP. Not used in the SENDTO service. Default: None
ADTLADDR	Local IP host address Indicates the local host internet address. this field is an unsigned four-byte integer value. The local host internet address is returned to the caller of the SENDTO service. Default: None
ADTRADDR	Remote IP host address Indicates the remote host internet address destination for the datagram being processed by the SENDTO service. This field is a unsigned four-byte integer value. Default: None
ADTLNAME	Local IP host name Indicates the local host internet name. This field is a 255-byte character string that is padded with blanks. The local host internet name is returned to the caller of the SENDTO service. Default: None
ADTRNAME	Remote IP host name Indicates the remote host internet name. this field is a 255-byte character string that is padded with blanks. It is only resolved through internal DNR calls and returned to the caller of the UDP data transfer service routines (RCVFROM and SENDTO) if the ADTOPTNS flag, ADTOPTNS-DDNR, is specified. This is to prevent the DNR call overhead on every UDP data transfer call. Default: None
ADTUCNDX	One word of user context Specifies one arbitrary word of user context to be associated with the connection. The information provided is not interpreted by CPT, and is merely saved with other connection information. Default: 0 (no user context)

200801-024040-200100

PARAMETER	DESCRIPTION
ADTOPTN1	Specifies data transfer options These are the ADT options that apply to UDP data transfer requests: ◆ ADTOPTN1_DODNR – Execute internal DNR calls during UDP data transfer service routine calls (RCVFROM and SENDTO) to resolve remote IP addresses into IP names in the ADTRNAME field. ◆ ADTOPTN1_NBLKR – Do not block on a call to the RCVFROM service (not used by the SENDTO service). Default: None **Note:** These options can be toggled on every UDP data transfer call even if the caller is using the same token.

Network Considerations

The ADT is a common data structure used for both client and server UDP applications. There are common and unique values specified for a particular service request.

This table describes network considerations for PL/1 API:

NAME	SERVER CONDITIONS FOR RCVFROM	CLIENT CONDITIONS FOR SENDTO
ADTLPORT	Local server well-known port selected by user application.	Local assigned transport provider port returned to user application.
ADTRPORT	Remote client transport provider port returned to user by user application.	Remote server transport provider well-known port selected by user application.
ADTSRVCE	Local server trans port provider service name selected by user application.	Remote server transport provider service name selected by user application.
ADTRADDR	Remote IP host address returned to user application.	Remote IP host address selected by or returned to user application. The client must specify this field or ADTRNAME.
ADTLNAME	Local IP host name returned to user application.	Local IP host name returned to user application.
ADTRNAME	Remote IP host name returned to user application only if ADTOPTN1_DDNR is specified in ADTOPTN1.	Remote IP host name selected by or returned to the user application. The client must specify this field or ADTRADDR. If ADTRADDR is used, ADTRNAME will only be returned if ADTOPTN1_DDNR is specified in ADTOPTN1.

2024040-200100

Return Codes

The SENDTO service return and diagnostic codes indicate the result of the execution. These values are in the ADTRTNCD and ADTDGNCD within the ADT. The diagnostic code is optional and generally indicates a transport provider return code. Read **Appendix C – MESSAGES AND CODES** in the ***CICS Programmer's Toolkit Installation and Administration Guide*** for the return code cross reference table.

This table describes the SENDTO service return codes:

RETURN	DIAGNOSTIC CODE	DESCRIPTION
CPTIRCOK		Successful.
CPTEVERS		Control block version number is not supported.
CPTETOKN		Specified token is not valid.
CPTEBUFF		Buffer address and/or length is invalid.
CPTEPRGE	Yes	CPT Interface is terminating.
CPTENAPI	Yes	Transport provider API is not available.
CPTETERM	Yes	Environment is being terminated.
CPTERLSE	Yes	Release indication.
CPTEDISC	Yes	Disconnect indication.
CPTEENVR	Yes	Transport provider API environment error.
CPTEFRMT	Yes	Transport provider API format error.
CPTEPROC	Yes	Transport provider API procedure error.
CPTABEND		Abnormal exception occurred.
CPTEOTHR	Yes	An undefined exception occurred.

200801-024040-200100

Example:

This example sends a message to a remote host. The return code is checked to determine SENDTO service completion status.

```
SAMP1: PROCEDURE OPTIONS (MAIN);

DCL    (T09FSNT)
       ENTRY OPTIONS (INTER ASSEMBLER);

DCL 1  CPT_ADT,
       %INCLUDE SYSLIB(T09KPADT);
       %INCLUDE SYSLIB(T09KPRCS);

DCL    MESSAGE CHAR (80)
   INIT('THIS IS THE PL/1 EXAMPLE');

%INCLUDE T09KPCON;

   /*
       Identify Service
   */

       ADTRPORT = 1580;
       ADTRNAME = '127.0.0.1';

   /*
       Application to process data
   */
   /*
       CPT SENDTO Facility Management service request
   */
       ADTBUFFA = ADDR(MESSAGE);
       ADTBUFFL = STG(MESSAGE);
       CALL T09FSNT (CPT_ADT);
       IF ADTRTNCD ^= 0
       THEN DO;
           /*
               Process and log SENDTO error
               Terminate WHILE condition
           */
           END;

   /*
       CPT Terminate Endpoint
   */

   /*
       Terminate transaction
   */
       EXEC   CICS RETURN;
```

TAKE

The TAKE service establishes ownership of a connection and associated internal CPT resources. The TAKE service is optional and does not affect an active connection nor does it issue any transport provider requests. This service affects CPT TRUE management routines scheduled on the user's behalf during task termination.

To invoke the TAKE service, a user application is required to first build an AFM and then to issue a call to the TAKE routine. The only valid and required arguments are the AFM version number and the connection token. On completion, a return code is set to indicate success or failure of the request.

This table shows the TAKE service arguments:

INCLUDE STATEMENT	SIZE	CREATED BY
T09KPAFM	34 (X'22')	User application or the LISTEN service

Structure

This is the structure of the TAKE service in PL/1 language:

```
DCL 1 CPT_AFM,
/*BEGIN %INCLUDE SYSLIB
(T09KPAFM)**********************************
/*************************************************************/
/*                          CPT_AFM                          */
/*   CICS PROGRAMMER'S TOOLKIT FACILITY MANAGEMENT CONTROL BLOCK
/*-----------------------------------------------------------*/
5  AFMVERS    FIXED BIN (15) INIT(2),    /* AFM BLOCK VERSION  */
5  AFMFUNC    FIXED BIN (15) INIT(0),    /* FUNCTION CODE      */
5  AFMTOKEN   POINTER  INIT(NULL),       /* DATA XFER TOKEN    */
5  AFMRSVD1   POINTER  INIT(NULL),       /* RESERVED           */
5  AFMRSVD2   FIXED BIN (31) INIT(0),    /* RESERVED           */
5  AFMRTNCD   FIXED BIN (31) INIT(0),    /* RETURN CODE        */
5  AFMDGNCD   FIXED BIN (31) INIT(0),    /* DIAGNOSTIC CODE    */
5  AFMRSVD3   FIXED BIN (31) INIT(0),    /* RESERVED           */
5  AFMRSVD4   FIXED BIN (15) INIT(0);    /* RESERVED           */
```

PARAMETER	DESCRIPTION
AFMVERS	Version Indicates the CPT version number of the argument list used by the calling program. This required field is set to a binary 2 for this release of CPT. Default: None
AFMFUNC	Function code Indicates the function or callable service ID requested by the application program. This field is not set by the application, but is initialized by the TRUE interface stub program. Default: None

200801-024040-200100

PARAMETER	DESCRIPTION
AFMTOKEN	Connection or endpoint token Specifies a token that represents a TCP connection, a TCP listening endpoint, or a UDP endpoint. A token is created by the TCP connection initiation routines or by the UDP data transfer and endpoint creation routines. The token is required. Default: 0
AFMRTNCD	Return code Indicates the return code set by the GIVE service. This value is returned and indicates the success or failure of the service. Default: 0
AFMDGNCD	Diagnostic code Indicates the diagnostic code received by the GIVE service for a transport provider request and is not set by the GIVE service. The GIVE service does not issue transport provider requests; hence, it never sets the diagnostic code. Default: 0

Completion Information

The TAKE service completes normally when all resources associated with this connection are processed.

On normal return to the application program, the general return code in AFMRTNCD is set to zero (CPTIRCOK). The diagnostic code in AFMDGNCD is always zero.

If the TAKE service completes abnormally, then some resources associated with this connection cannot be successfully transferred from one task to another. The general return code in AFMRTNCD and the diagnostic code in AFMDGNCD indicate the nature of the failure. The diagnostic code (AFMDGNCD) is not used by the TAKE service and no information is returned.

Return Codes

The TAKE service return code indicates the result of the execution. This value is in the AFMRTNCD within the AFM. Read **Appendix C – MESSAGES AND CODES** in the ***CICS Programmer's Toolkit Installation and Administration Guide*** for the return code cross reference table.

This table describes the return codes for the TAKE service:

RETURN	DIAGNOSTIC CODE	DESCRIPTION
CPTIRCOK		Successful.
CPTEVERS		Control block version number is not supported.
CPTETOKN		Specified token is not valid.
CPTENAPI		Transport provider API is not available.
CPTABEND		Abnormal exception occurred.
CPTEOTHR		An undefined exception occurred.

200801-024040-200100

Usage Information

The TAKE service acquires ownership of a connection from one task to another. This service is non-blocking and does not effect any pending transport provider data transfer requests. The association established by the TAKE service lets the CPT properly manage resources during task termination. This ability to GIVE and TAKE ownership of connections offers a range of programming options, while still providing CPT with resource management capabilities.

The TAKE service requires the application to set the AFM version number and token fields. No other AFM fields are referenced.

When a connection is established there are internal CPT resources associated with that connection. CPT is responsible for proper clean-up of those resources on task or transaction termination. These resources include storage allocated by CPT, the API, and the transport provider storage.

A server application is a good example of how the TAKE service benefits a user application. A listening task issues the GIVE service and starts a new transaction to handle data transfer. The data transfer transaction then TAKEs the connection. This sequence prevents a connection from being closed (implicitly by the CPT task termination exit) if the server application terminates. However, if the data transfer transaction terminates without issuing an explicit close (CPT CLOSE service), an implicit close is scheduled and resource management is handled by the CPT task termination exit.

Additionally, an implicit TAKE facility is implemented with the SEND, RECEIVE, and TRANSLATE services. Any task that issues a SEND, RECEIVE, or TRANSLATE service obtains control of the connection and associated resources. The advantage is that the TAKE service is optional. Even though TAKE can be implicit and therefore optional, Interlink recommends that you issue TAKE to avoid having a GIVE connection not associated with any transactions. Ownership of a connection and resources provide for clean-up processing during abnormal termination.

The version number (AFMVERS) indicates the CPT release level in which this user application program is written. This required field is set to a binary 2 and is validated by the TAKE service before processing the request.

The function code (AFMFUNC) indicates the CPT callable service ID. The field is not initialized by a user application program and has little value to the application except for dump analysis. The function code identifies and maps an argument list with the error or trace log and dump analysis.

The token (AFMTOKEN) indicates the connection and internal resources to be processed by the TAKE service. This is a required field and is validated by the TAKE service before processing the request.

The AFMOPTCD field specifies TAKE service processing control options, and provides a mechanism for event notification on return to the application program. Currently, this field is not used by application programs.

Example:

In this example a data processing transaction retrieves the ACM, issues the TAKE service, and then performs some required operation. The token is loaded from the ACM and is used by the TAKE service. The return code is checked to determine TAKE service completion status.

```
SAMP1: PROCEDURE OPTIONS (MAIN);

DCL    (T09FTAK)
       ENTRY OPTIONS (INTER ASSEMBLER);
DCL    (CPTPRT,MSGPRT) POINTER;

DCL 1  CPT_ACM,
       %INCLUDE SYSLIB(T09KPACM);
DCL 1  CPT_AFM,
       %INCLUDE SYSLIB(T09KPAFM);
DCL 1  CPT_RCS,
       %INCLUDE SYSLIB(T09KPRCS);

%INCLUDE T09KPCON;

   /*
      Retrieve arguments
    */
      CICS RETRIEVE FROM(CPT_ACM) LENGTH(STG(CPT_ACM));

      AFMTOKEN = ACMTOKEN;

   /*
      CPT TAKE Facility Management service request
    */
      CALL T09FTAK (CPT_AFM);
      IF AFMRTNCD ^= 0
      THEN DO;
      /*
         Process and log TAKE error
         Terminate transaction
       */
      END;
   /*
      Application and CPT Data Transfer (SEND/RECV) service request
    *
   /*
      CPT Connection Release service request
    */
   /*
      Terminate transaction
    */
      EXEC   CICS RETURN;
```

200801-024040-200100

Example:

In this example a data processing transaction retrieves the ACM and then performs some required data transfer operation. The token is loaded from the ACM and is used for CPT service requests. The difference between this example and the first example is that no TAKE service is explicitly issued by the application. The CPT data transfer services, SEND and RECEIVE, implement the TAKE service internally, so an explicit TAKE service request can be omitted.

```
SAMP1: PROCEDURE OPTIONS (MAIN);

DCL    (T09TFREC,T09FSEN,T09FCLO)
       ENTRY OPTIONS (INTER ASSEMBLER);DCL (CPTPRT,MSGPRT) POINTER;

DCL 1  CPT_ACM,
       %INCLUDE SYSLIB(T09KPACM);
DCL 1  CPT_ADT,
       %INCLUDE SYSLIB(T09KPADT);
DCL 1  CPT_ACL,
       %INCLUDE SYSLIB(T09KPACL);
DCL 1  CPT_RCS,
       %INCLUDE SYSLIB(T09KPRCS);

%INCLUDE T09KPCON;
   /*
      Retrieve arguments
    */
      CICS RETRIEVE FROM(CPT_ACM) LENGTH(STG(CPT_ACM));

      AFMTOKEN = ACMTOKEN;
   /*
      Application and CPT Data Transfer (SEND/RECV) service request
    *
   /*
      CPT Connection Release service request
    */
   /*
      Terminate transaction
    */
      EXEC   CICS RETURN;
```

TRANSLATE

The TRANSLATE service translates data between EBCDIC and ASCII character sets. CPT is customized with a default translation table; however, applications can override the default. The TRANSLATE service does not affect an active connection nor issue any transport provider requests.

To invoke the TRANSLATE service, a user application is required to first build an Argument for Data Translation (AXL) and then to issue a call to the TRANSLATE routine. The AXL is required to contain the version number, connection token, user buffer address and length, and type or direction of translation requested. Additional arguments for application specific translation tables are supported. When the TRANSLATE service completes, the buffer contents are converted into the corresponding characters and a return code is generated indicating the status of the request.

This table describes the TRANSLATE service arguments:

INCLUDE STATEMENT	SIZE	CREATED BY
T09KPAXL	32 (X'20')	User application

Structure

This is the structure of the TRANSLATE service in PL/1 language:

```
DCL 1 CPT_AXL,
/*BEGIN %INCLUDE SYSLIB
(T09KPAXL)**********************************
/*************************************************************/
/*                        CPT_AXL                            */
/*         CICS PROGRAMMER'S TOOLKIT TRANSLATION BLOCK       */
/*-----------------------------------------------------------*/
5  AXLVERS    FIXED BIN (15) INIT(2),    /* AXL BLOCK VERSION   */
5  AXLFUNC    FIXED BIN (15) INIT(0),    /* FUNCTION CODE       */
5  AXLTOKEN   POINTER  INIT(NULL),       /* DATA XFER TOKEN     */
5  AXLSADDR   POINTER  INIT(NULL),       /* SOURCE TEXT ADDRSS  */
5  AXLSLENG   FIXED BIN (31) INIT(0),    /* SOURCE TEXT LENGTH  */
5  AXLRTNCD   FIXED BIN (31) INIT(0),    /* RETURN CODE         */
5  AXLDGNCD   FIXED BIN (31) INIT(0),    /* DIAGNOSTIC CODE     */
5  AXLXMRSV    CHAR (1) INIT(LOW(1)),    /* RESERVED FOR MODE   */
5  AXLXMODE  BIT (8) INIT('00000000'B), /* CHARACTER SET MODE  */
5  AXLXTRSV    CHAR (1) INIT(LOW(1)),    /* RESERVED FOR TYPE   */
5  AXLXTYPE  BIT (8) INIT('00000000'B), /* TRANSLATION TYPE    */
5  AXLTABLE   POINTER  INIT(NULL);       /* USER XLATE TABLE    */
                                         /*   ADDRESS, IF ANY   */
/*END   %INCLUDE SYSLIB
(T09KPAXL)**********************************
```

The following constants are contained in the T09MAC library member, T09KPCON:

```
             /*-------------------------------------------*/
             /*       TRANSLATION TYPE REQUEST            */
             /*-------------------------------------------*/
AXLXTYPE_ATOE BIT (8) INIT('00000001'B), /* ASCII-TO-EBCDIC */
AXLXTYPE_ETOA BIT (8) INIT('00000010'B), /* EBCDIC-TO-ASCII */
```

200801-024040-200100

```
AXLXTYPE_AUPC BIT (8) INIT('00000100'B), /* A TO UPPERCASE A*/
AXLXTYPE_EUPC BIT (8) INIT('00001000'B), /* E TO UPPERCASE E*/

             /*-----------------------------------------*/
             /*          CHARACTER SET MODE             */
             /*-----------------------------------------*/
AXLXMODE_SBCS BIT (8) INIT('00000000'B), /* SINGLE BYTE C.S. */
AXLXMODE_DBCS BIT (8) INIT('00000001'B), /* DOUBLE BYTE C.S. */
AXLXMODE_MIXD BIT (8) INIT('00000010'B), /* MIXED SBCS/DBCS */
AXLXMODE_NUMS BIT (8) INIT('00000100'B) /* NUMBER SET       */
```

PARAMETER	DESCRIPTION
AXLVERS	Version Indicates the CPT version number of the argument used by the calling program. This required field is set to a binary 2 for this release of CPT. Default: None
AXLFUNC	Function code Indicates the function or callable service ID requested by the application program. This field is not set by the application, but is initialized by the TRUE interface stub program. Default: None
AXLTOKEN	Connection token Specifies the token that represents the connection. A token is obtained from a connection management request. The token is required. Default: 0
AXLSADDR	Source text buffer address Indicates the address of the user data buffer to be translated. This required field is a contiguous segment of storage accessible to the user task. The storage area can be aligned on any boundary convenient to the application program. Default: 0
AXLSLENG	Source text buffer length Indicates the length (in bytes) of user data buffer in the storage area, as identified by the AXLSADDR field. This is a required field. A zero value causes the request to fail. Default: 0
AXLRTNCD	Return code Indicates the return code set by the TRANSLATE service. This value is returned and indicates the success or failure of the service. Default: 0
AXLDGNCD	Diagnostic code Indicates the diagnostic code set by the service request. This value specifies a unique number associated with the return code and identifies the translation error. Default: 0

20-024040-200100

PARAMETER	DESCRIPTION
AXLXMODE	Specifies TRANSLATE service translation mode or character set. ◆ AXLXMODE-SBCS – Indicates single-byte character set translation. ◆ AXLXMODE-DBCS – Indicates double-byte character set translation. This option is currently not supported. ◆ AXLXMODE-MIXD – Indicates mixed mode character set translation. This mode specifies single- and double-byte translation. This option is currently not supported. ◆ AXLXMODE-NUMS – Indicates numeric set translation. This option is currently not supported. Default: AXLXMODE-SBCS
AXLXTYPE	Specifies TRANSLATE service translation type or direction. ◆ AXLXTYPE-ATOE – Indicates ASCII to EBCDIC translation. ◆ AXLXTYPE-ETOA – Indicates EBCDIC to ASCII translation. ◆ AXLXTYPE-AUPC – Indicates ASCII to uppercase ASCII translation. ◆ AXLXTYPE-EUPC – Indicates EBCDIC to uppercase EBCDIC. translation. Default: None
AXLTABLE	Address of user translation table. Default: None

Completion Information

The TRANSLATE service completes normally when the data is translated into the corresponding character set representation.

On normal return to the application program, the general return code in AXLRTNCD is set to zero (CPTIRCOK). The diagnostic code in general AXLDGNCD is set to zero.

If the TRANSLATE service completes abnormally an error associated with translation occurred. The general return code in AXLRTNCD and the diagnostic code in AXLDGNCD indicate the nature of the failure.

200801-024040-200100

Return Codes

The TRANSLATE service return and diagnostic codes indicate the result of the execution. These values are in the AXLRTNCD and AXLDGNCD within the AXL. Read **Appendix C – MESSAGES AND CODES** in the ***CICS Programmer's Toolkit Installation and Administration Guide*** for the return code cross reference table.

This table describes the TRANSLATE service return codes:

RETURN	DIAGNOSTIC CODE	DESCRIPTION
CPTIRCOK		Successful.
CPTEVERS		Control block version number is not supported.
CPTETOKN		Specified token is not valid.
CPTEBUFF		Buffer address and/or length is invalid.
CPTECHAR		Translation table character set is Invalid.
CPTEMODE		Translate mode specification is invalid.
CPTEFRMT		Format or specification error.
CPTENAPI		Transport provider API is not available.
CPTABEND		Abnormal exception occurred.
CPTEOTHR		An undefined exception occurred.

Usage Information

The TRANSLATE service translates data between EBCDIC and ASCII. The requirement for translation is application dependent.

The version number (AXLVERS) indicates the CPT release level in which this user application program is written. This required field is set to a binary 2 and is validated by the TRANSLATE service before processing the request.

The function code (AXLFUNC) indicates the CPT callable service ID. The field is not initialized by a user application program and has little value to the application except for dump analysis. The function code can identify and map an argument list with the error or trace log and dump analysis.

The token (AXLTOKEN) indicates the connection associated with this translation request. This field is required; however, no transport provider requests are issued. The token is used for internal logging support requirements.This required field is validated by the TRANSLATE service before processing the request.

The AXLXMODE field specifies the character set mode. This field sets single-byte, double-byte, or mixed character set translation. Currently, only single-byte character set translation is supported (and is the default).

024040-200100

The AXLXTYPE field specifies the translation direction. This required field indicates EBCDIC to ASCII, or ASCII to EBCDIC. Additionally, characters can be transacted into the corresponding uppercase values.

200801-024040-200100

Example:

In this example an established connection sends data to an ASCII host. The data is translated by the application and then is sent over the connection. The version number, token, data buffer address and length, and translation type are set in the AXL. The default translation mode of SBCS is selected. The return code is checked to determine TRANSLATE service completion status.

```
SAMP1: PROCEDURE OPTIONS (MAIN);

DCL    (T09FXLA)
       ENTRY OPTIONS (INTER ASSEMBLER);
DCL    (CPTPRT,MSGPRT) POINTER;

DCL 1  CPT_AXL,
       %INCLUDE SYSLIB(T09KPAXL);
DCL 1  CPT_RCS,
       %INCLUDE SYSLIB(T09KPRCS);

DCL DATA BIT (1) INTI ('1'B);

DCL    MESSAGE CHAR (80)
       INIT('WELCOME TO THE CICS/API PROGRAMMING WORLD');

%INCLUDE T09KPCON;

   /*
      CPT Connection Management service request
    */

      ADTTOKEN = ACMTOKEN;

   DO WHILE (DATA);

      /*
         Application to process data
       */

      /*
         CPT TRANSLATE service request
       */
            AXLBUFFA = ADDR(BUFFER);
            AXLBUFFL = 80;
            AXLXTYPE = AXLXTYPE_ETOA;
            CALL T09FXLAT (CPT_AXL);
            IF AXLRTNCD ^= 0
         THEN DO;
         /*
            Process and log TRANSLATE error
            Terminate WHILE condition
          */
         END;

      /*
         CPT SEND service request
       */

   END;
   /*
      CPT Connection Release service request
    */
   /*
      Terminate transaction
    */
      EXEC   CICS RETURN;
```

Example:

In this example an established connection receives ASCII data. The data is translated from ASCII to EBCDIC before application processing. The token, data buffer address and length, and translation type are set in the AXL. The default translation mode of SBCS is selected. The return code is checked to determine TRANSLATE service completion status.

```
SAMP2: PROCEDURE OPTIONS (MAIN);

DCL    (T09FXLA)
       ENTRY OPTIONS (INTER ASSEMBLER);
DCL    (CPTPRT,MSGPRT) POINTER;

DCL 1  CPT_AXL,
       %INCLUDE SYSLIB(T09KPAXL);
DCL 1  CPT_RCS,
       %INCLUDE SYSLIB(T09KPRCS);

DCL DATA BIT (1) INIT ('1'B);

DCL    MESSAGE CHAR (80);

%INCLUDE T09KPCON;

   /*
      CPT Connection Management service request
    */

      ADTTOKEN = ACMTOKEN;

   DO WHILE (DATA);
      /*
         CPT RECEIVE service request
       */
      /*
         CPT TRANSLATE service request
       */
            AXLBUFFA = ADTBUFFA;AXLBUFFL = ADTBUFFL;
            AXLXTYPE = AXLXTYPE_ATOE;
            CALL T09FXLAT (CPT_AXL);
            IF AXLRTNCD ^= 0
         THEN DO;
         /*
            Process and log TRANSLATE error
            Terminate WHILE condition
          */
         END;
      /*
         Application to process data
       */
    END;
    /*
       CPT Connection Release service request
     */
    /*
       Terminate transaction
     */
       EXEC   CICS RETURN;
```

Return Codes

The T09KPRCS copy member resolves API service return codes.

```
/*************************************************************/
/*     EXCEPTION CODES POSTED IN RETURN CODE ARGUMENT FIELDS*/
/*-----------------------------------------------------------*/
      /* REQUEST COMPLETED SUCCESSFULLY          */
DCL CPTIRCOK FIXED BIN (31)  INIT(0);
/*************************************************************/
/*                        WARNINGS                           */
/*-----------------------------------------------------------*/
      /* TIME RECEIVED SERVICE CALL TIMED OUT
DCL CPTWTIMO FIXED BIN (31) INIT(0);
      /* BUFFER, QUEUE SIZES RESET TO SYSTEM LIMITS*/       /
DCL CPTWNEGO FIXED BIN (31)  INIT(4);                       /
      /* RECEIVE WOULD BLOCK (NO DATA CURRENTLY AVAIL.)
DCL CPTWBLCK FIXED BIN (31) INIT(6);
      /* UDP - THIS IS NOT THE WHOLE DATAGRAM
DCL CPTWNEOM FIXED BIN (31) INIT(8);
      /* SEP TYPE RECEIVE FOUND NO SEPARATOP CHARACTERS
DCL CPTWNSEP FIXED BIN (31) INIT(10);
      /* OTHER WARNING                           */
DCL CPTWEXCP FIXED BIN (31) INIT(15);
/*************************************************************/
/*              CONTROL BLOCK ARGUMENT ERRORS                */
/*-----------------------------------------------------------*/
      /* CONTROL BLOCK VERSION NUMBER NOT SUPPORTED*/
DCL CPTEVERS FIXED BIN (31) INIT(17);
      /* REQ HOST/SERVICE/PORT CONNECTION NOT FOUND*/
DCL CPTECONN FIXED BIN (31) INIT(18);
      /* SPECIFIED PROTOCOL NOT SUPPORTED        */
DCL CPTEPROT FIXED BIN (31) INIT(19);
      /* SPECIFIED DATA TRANSFER TOKEN IS INVALID*/
DCL CPTETOKN FIXED BIN (31) INIT(20);
      /* BUFFER ADDRESS AND/OR LENGTH INVALID    */
DCL CPTEBUFF FIXED BIN (31) INIT(21);
      /* TRANSLATE CHARACTER SET IS INVALID      */
DCL CPTECHAR FIXED BIN (31) INIT(22);
      /* TRANSLATE MODE SPECIFICATION IS INVALID */
DCL CPTEMODE FIXED BIN (31) INIT(23);
      /* CLOSE MODE SPECIFICATION IS INVALID     */
DCL CPTECOPT FIXED BIN (31) INIT(24);
      /* SPECIFIED TRANSLATE TABLE NOT CORRECT   */
DCL CPTETABL FIXED BIN (31) INIT(25);
      /* DESIGNATED TRANSACTION ID CANNOT START  */
DCL CPTETRID FIXED BIN (31) INIT(26);
      /* OTHER TPL FORMAT OR SPECIFICATION ERROR */
DCL CPTEFRMT FIXED BIN (31) INIT(31);
      /* RECV TIME OUT VALUE MUST BE > ZERO
DCL CPTETIME FIXED BIN (31) INIT(27);
      /* RECV TYPE SEP REQUIRES # SEP CHARACTERS = 1 OR 2
DCL CPTESEP# FIXED BIN (31) INIT(28);
      /* RECV OPTIONS COMBINATION IS INVALID
DCL CPTEOPTN FIXED BIN (31) INIT(29);
      /* RECV OPTION NOT SUPPORTED BY TRANSPORT CARRIER
DCL CPTEOPRL FIXED BIN (31) INIT(30);
      /* OTHER TPL FORMAT OR SPECIFICATION ERROR
```

```
DCL CPTEFRMT FIXED BIN (31) INIT(31);
/*************************************************************/
/*              LOCAL ENVIRONMENT ERRORS                     */
/*-----------------------------------------------------------*/
      /* SELECTED PORT IS BUSY WITH ACTIVE SERVER */
DCL CPTEPBSY FIXED BIN (31) INIT(33);
      /* SNS/API NOT FULLY AVAILABLE, RETRY       */
DCL CPTENAPI FIXED BIN (31) INIT(34);
      /* REQUESTED FACILITY IS NOT AVAILABLE      */
DCL CPTENAVL FIXED BIN (31) INIT(35);
      /* ENVIRONMENT IS BEING DRAINED             */
DCL CPTEDRAN FIXED BIN (31) INIT(36);
      /* ENVIRONMENT IS BEING TERMINATED          */
DCL CPTETERM FIXED BIN (31) INIT(40);
      /* OTHER TPL ENVIRONMENTAL CONDITION        */
DCL CPTEENVR FIXED BIN (31) INIT(47);
/*************************************************************/
/*               CONNECTION ERRORS                           */
/*-----------------------------------------------------------*/
      /* ORDERLY RELEASE OF REMOTE CONNECTION REQ */
DCL CPTERLSE FIXED BIN (31) INIT(65);
      /* REMOTE CONNECTION NOT AVAILABLE OR ABORTED*/
DCL CPTEDISC FIXED BIN (31) INIT(68);
      /* REMOTE CONNECTION ENVIRONMENT TERMINATING*/
DCL CPTEPRGE FIXED BIN (31) INIT(72);
      /* OTHER TPL CONNECTION/DATA INTEGRITY ERROR*/
DCL CPTEINTG FIXED BIN (31) INIT(79);
/*************************************************************/
/*               OTHER EXCEPTIONS                            */
/*-----------------------------------------------------------*/
      /* PROCEDURAL ERROR                         */
DCL CPTEPROC FIXED BIN (31) INIT(143);
      /* ABNORMAL TERMINATION                     */
DCL CPTABEND FIXED BIN (31) INIT(254);
      /* OTHER ERROR                              */
DCL CPTEOTHR FIXED BIN (31) INIT(255);
```

PARAMETER	DESCRIPTION
CPTIRCOK	Request completed successfully Indicates the requested service completed successfully.
CPTWNEGO	System limits applied to buffer and queue sizes A connection management service returned modified API buffering values. The ACMQSEND, ACMMSEND, ACMQRECV or ACMMRECV values were modified during transport provider connection negotiation. This return code does not generally indicate a serious error, but rather a warning.
CPTWBLCK	RECEIVE would block (no data currently available) No data was available on a TCP connection if the call was to the RECEIVE service, or no datagrams were available at a UDP endpoint if the call was to the RCVFROM service. In both cases, ADTOPTN1 was set to ADTOPTN1_NBLKR
CPTWNEOM	This is not the whole datagram On a RCVFROM service call, ADTBUFFA for a length of the ADTBUFFL was not large enough to hold the entire datagram. Subsequent RCVFROM calls will be required to retrieve the rest of the datagram.

200801-024040-200100

PARAMETER	DESCRIPTION
CPTWNSEP	On a RECEIVE service call with the SEP option specified (ADTOPTN1_TYPSP) no indicated separator sequence was found in the data. Check user data and make sure the ADTTIMEO value was long enough to receive all of the data.
CPTWEXCP	Transport provider exceptional condition A transport provider exceptional error occurred. Review diagnostic code for additional information.
CPTEVERS	Argument version number not supported Indicates the version number specified is not supported. The service request is not executed
CPTECONN	Requested host/service port connection not found A connection management service request failed due to an invalid or unresolved host, service name or port number specification. The service request is not executed.
CPTEPROT	Specified protocol not supported A connection management service request failed due to an invalid or unsupported protocol specification. The protocol specified was not TCP or UDP. The service request is not executed.
CPTETOKN	Specified token is invalid Specified token in service argument is invalid. The service request is not executed.
CPTEBUFF	Buffer address and/or length invalid Specified buffer address and/or length is invalid. A data transfer or data translation service request failed verification of buffer address or length. The service request is not executed.
CPTCHAR	Translate character set is invalid Specified translation character set is invalid. The translation service request is not executed.
CPTEMODE	Translate mode specification is invalid Specified translation mode is invalid. The Translation service request is not executed.
CPTECOPT	CLOSE mode specification is invalid An invalid or unknown CLOSE option (ACLOPTNS) was specified on a call to the CLOSE service.
CPTETABL	Specified translate table not correct An invalid character set table was specified on a call to the TRANSLATE service.
CPTETRID	Designated transaction ID cannot be started The LISTEN service attempted to start a transaction that failed. The LISTEN service request is terminated and the well-known server port is released.

J24040-200100

PARAMETER	DESCRIPTION
CPTETIME	RECEIVE service timeout value must be greater than zero Specifying any of these options on a RECEIVE call with an ADTTIMEO=0 will result in CPTETIME being returned in ADTRTNCD: ADTOPTN1_TYPLL ADTOPTN1_TYPSP ADTOPTN1_TMRCV ADTOPTN1_TMPRT
CPTESEP#	RECEIVE type SEP requires # SEP chars = 1 or 2 On a RECEIVE service call with the SEP option specified (ADTOPTN1_TYPSP), ADTSEP# was not equal to 1 or 2.
CPTEOPTN	RECEIVE options combination is invalid On a RECEIVE service call, more than one of these options was specified: ADTOPTN1_NBLKR ADTOPTN1_TYPLL ADTOPTN1_TYPSP ADTOPTN1_TMRCV ADTOPTN1_TMPRT
CPTEOPRL	RECEIVE option not supported by transport carrier You have attempted a CPT function that is not supported by the release of SNS/TCPaccess that you are running.
CPTEFRMT	Other transport provider format or specification error A transport provider format or specification error occurred. Review diagnostic code for additional information. The service request failed to execute successfully.
CPTEPBSY	Selected port is busy with active server The LISTEN service attempted to establish a connection to the local transport provider but the specified port was already allocated. The LISTEN service request failed to execute.
CPTENAPI	API, transport provider or CICS not available The transport provider, API communication subsystem or CICS is not available. Review diagnostic code for additional information. The service request failed to execute.
CPTEDRAN	Transport provider or API is being drained The transport provider or API communication subsystem is being drained. This is an indication that the CPT programming interface may be terminating.
CPTETERM	Transport provider or API is being terminated The transport provider or API communication subsystem is being terminated. This is an indication that the CPT programming interface is terminated. All connections have been closed and no communication is available.
CPTEENVR	Other transport provider environment error A transport provider environment error occurred. Review diagnostic code for additional information. The service request failed to execute successfully.
CPTERLSE	Orderly release of full duplex connection indicated Indicates that the remote host has released its half of the full duplex connection.The connection is still available for communication, but the user application will not receive data for the remote host.

200801-024040-200100

PARAMETER	DESCRIPTION
CPTEDISC	Remote connection not available or aborted Indicates that a connection request failed or an established connection was aborted. The connection request is not established or an established connection is terminated.
CPTEPRGE	CICS programmer's toolkit interface terminated The CPT interface has been terminated. The CPT termination transaction has been initiated by either the termination transaction or CICS shutdown PLT entry.
CPTEINTG	Other transport provider integrity error A transport provider integrity error occurred. Review diagnostic code for additional information. The service request failed to execute successfully.
CPTEPROC	Other Transport provider procedure error A transport provider procedure error occurred. Review diagnostic code for additional information. The service request failed to execute successfully.
CPTABEND	Abnormal environmental error The service request experienced an abnormal termination condition. The service request failed to execute. The CICS dump data set contains additional information related to the abend.
CPTEOTHR	Any other currently undefined condition An unknown condition was detected. Review the diagnostic code for additional information. The service request failed to execute.

CICS Storage Requirements

All CICS storage is shared and allocated from above the 16M line, if possible. Storage requirements for a TCP connection or a UDP endpoint within the CICS address space are:

- `1080 + ((72 + xxxMSEND) * xxxQSEND) + ((72 + xxxMRECV) * xxxQRECV)`

  where `xxx` is either ACM for TCP connections or ADT for UDP endpoints.

Note: For TCP server applications, an additional 1080-byte overhead is required to allocate the listening token/socket. This computation accounts only for the TCP connection that is subsequently established by a listening tocken/socket or by the storage for a UDP endpoint.

Note: The send buffer queue is allocated only if an application calls the `SEND` or `SENDTO` services. Determining the `xxxMSEND` and `xxxQSEND` values is explained below. The receive buffer queue is allocated only if an application calls the `RECEIVE` or `RCVFRM` services. Determining the `xxxMRECV` and `xxxQRECV` values is explained below.

Note: The receive buffer queue is allocated only if an application calls the `RECEIVE` or `RECVFROM` services. Determining the `xxxMRECV` and `xxxQRECV` values is explained below.

The maximum send and receive data sizes, the maximum send and receive buffer sizes, and the send and receive queues' sizes (maximum number of outstanding requests per endpoint) are determined initially by the `ACPCONFG` macro, `ACFTIB`. Read the ***SNS/TCPaccess Customization Guide***, pages A-70 and A-71, for information. The TCP maximum data size default that can be sent or received is 32K. The maximum data buffer size default is 64K. CPT negotiates with SNS/PCaccess to determine these values:

- Determine final `xxxQSEND` value:

```
If xxxSEND is not specified,
      use the lesser of 4 or the value in DFQSEND
   else
      use the lesser of the value specified in xxxQSEND or
      DFQSEND.
```

- Determine final `xxxQRECV` value:

```
If xxxQRECV is not specified,
      use the lesser of 4 or the value in DFGRECV
   else
      use the lesser of the value specified in xxxQRECV or
      DFQRECV.
```

- Determine final `xxxMSEND` value:

```
If xxxMSEND is not specified,
      use the lesser of 4096 or the value in MXLTSND
   else
```

```
        use the lesser of the value specified xxxMSEND or
        MXLTSND
either of which must be less than the following
computation:
        (the lesser of DFLSEND or 61440) / the final
        xxxQSEND value
    else
        use the quotient from this computation as thefinal
        xxxMSEND value.
```

- **Determine final xxxMRECV value:**

```
If xxxMRECV is not specified,
        use the lesser of 4096 or the value in MXLTRCV
    else
        use the lesser of the value specified in xxxMSEND or
        MXLTRCV
either of which must be less than the following
computation:
(the lesser of DFLRECV or 61440) / the final xxxQRECV value
    else
        use the quotient from this computation as the final
        xxxMRECV value.
```

Glossary

This glossary defines the terms and acronyms referred to in Interlink documentation. The first section provides definitions of terms. The second section is a quick reference list of acronyms and what they stand for. All acronyms included in the second section are defined in the first section.

Definitions

Abnormal End of Task (ABEND)

Termination of a task before its completion because of an error condition that cannot be resolved by recovery facilities while the task is executing.

Access Control Facility 2 (ACF2)

A facility that provides security and access control on the mainframe for a user.

Access Control ID (ACID)

An alphanumeric value up to eight characters in length that TOP SECRET uses to determine a user's access to system resources. Associated with each unique ACID is a set of resource access authorizations and/or administrative authorizations.

Access Method

A mainframe data management routine that moves data between storage and an I/O device in response to requests made by a program.

Access Method Control Block (ACB)

A control block that links an application program to VSAM or VTAM.

Access Method Services (AMS or IDCAMS)

A multifunction service program that defines VSAM data sets and allocates space for them, converts sequential data sets to key-sequenced data sets, modifies data set attributes in the catalog, reorganizes data sets, facilitates data portability between operating systems, creates backup copies of data sets and indexes, helps make inaccessible data sets accessible, and lists the records of data sets and catalogs.

200801-020080-310100

Access Module Control Block (AMDCB)

A control block used by the NFS server to hold all access method dependent data for the duration of the file usage.

Accessor Environment Element (ACEE)

A description of the correct user, including User ID, current connect group, user attributes, and group attributes. An ACEE is constructed during user identification and verification.

Address Resolution Protocol (ARP)

A broadcast-based method for dynamically translating between IP addresses and physical addresses.

Advanced Function Printing (AFP)

A collection of software and hardware products that provides for quality printing from data processing systems. Facilities include electronic forms, text and graphics merging, bar codes, images, signatures, logos, etc.

Advanced Peer-to-Peer Communication (APPC)

An IBM protocol that is part of the SNA architecture that enables structured conversations between two networked applications. Also referred to as a Logical Unit (LU) Type 6.2. (See **Logical Unit.**)

Advanced Peer-to-Peer Networking (APPN)

An extension to the SNA networking architecture that supplies networking functions that are easy to use, are dynamic, and give control of the network to the peer systems that make up the network. APPN lets peer systems participate in and control a network that does not require the control of a traditional SNA host. APPN also handles routing of data by intermediate systems in the network so that all systems become logically adjacent, whether or not they are physically adjacent.

Advanced Resolution Protocol (ARP)

A broadcast-based method for dynamically translating between IP addresses and physical addresses.

American National Standards Institute (ANSI)

An organization that provides a standard specification for languages such as C and COBOL so that systems and programs can be developed to be compatible across platform boundaries.

American Standard Code for Information Interchange (ASCII)

A standard character code used in many computer systems. Traditional ASCII uses only seven bits out of eight possible to represent data, but extended ASCII uses all eight bits to define additional characters.

Application Program Control Block (APCB)

A data structure supplied by the application program as the primary anchor for information required to execute subsequent requests.

Application Program Interface (API)

The formally-defined programming language interface between an IBM system control program or program product and its user.

Architecture

The specific components of a computing system and the way they interact with one another.

ARPANET

A network created by the Defense Advanced Research Projects Agency (DARPA) that connects approximately 150 sites at universities and corporations doing research for the U.S. government. The ARPANET uses the TCP/IP protocol suite. It is part of the Internet.

Attribute Database (ADB)

A catalog database used by the NFS server to hold names of files allocated within the UFS.

20-020080-310100

Authorized Program Facility (APF)

A security feature of MVS that controls access to authorized system libraries. Programs in these libraries can enter a system-privileged state that permits greater control of internal system processes. These data sets must be pre-defined in order to be used.

Basic Mapping Support (BMS)

A facility that provides CICS commands and options that can be used to format data between an application program and a device in a standard manner.

Berkeley Software Distribution (BSD)

A series of UNIX system implementations developed at the University of California, Berkeley.

Big-endian

A format for storage or transmission of binary data in which the most significant bit (or byte) comes first. The reverse convention is called little-endian.

Binding

A process of associating a variable with an absolute address, identifier, or virtual address, or with a symbolic address or label in a program. (See **Unbinding**.)

Booting

The process of powering up the computer, testing to determine which attached hardware devices are running, and bringing the operating system kernel into memory and operation.

Build a Directory Entry List (BLDL)

A system service call that obtains a list of information from the directory of a Partitioned Data Set (PDS) or Partitioned Data Set Extended (PDSE).

Bulk Data Transfer (BDT)

A Courier-defined protocol for transferring large amounts of data between a client and server.

CA-TOP SECRET

A security program developed by Computer Associates that validates all user ID and password combinations for sign-on security and all file accesses for data set security.

Caltech Intermediate Form (CIF)

Describes an output file that contains data produced by `cifplot`.

Catalog Entry

A data set containing extensive information required to locate other data sets, to allocate and de-allocate storage space, to verify the access authority of a program or operator, and to accumulate data set usage statistics.

CICS Programmer's Toolkit (CPT)

An application programming interface between CICS application programs and communication subsystems, based on open network protocols.

Client

In a Sun ONC/NFS environment, a machine that uses the resources of the network.

Connection

A communications mechanism or program association that enables interaction between two functions or systems.

Connection Endpoint (CEP)

The termination of one end of a connection within a service access point.

Connection ID (CID)

An eight-bit data item used by VTAM to uniquely identify a session between to Logical Units (LUs).

Connection-Oriented Transport Service (COTS)

A service that provides the means to establish, maintain, and release transport connections providing duplex transmission between two transport users.

200801-020080-310100

Connectionless Network Protocol (CLNP)

The ISO standard for providing connectionless service; the functional equivalent of DARPA IP.

Connectionless Network Service (CLNS)

A service that uses UDP as the transport mechanism.

Connectionless Transport Protocol (CLTP)

Established a TCP connection without going through a specific host.

Connectionless Transport Service (CLTS)

Connectionless-mode service within the transport layer.

Control Vector

In SNA, an indexed data item that accompanies a session control Request Unit (RU). Control vectors can take many different forms, based on the information they contain.

Customer Information Control System (CICS)

An IBM mainframe control system combining elements of database management and data communications, intended to handle transaction-oriented applications. CICS enables transactions entered at remote terminals to be processed concurrently by user-written application programs. It includes facilities for building, using, and maintaining databases. CICS uses VTAM for communication with devices and programs in the network and VSAM for accessing data sets.

Daemon

A process that handles system-wide functions, such as network administration, or line printer spooling operations. It is a background task that waits for a request for service from a particular port.

Data Control Block (DCB)

A memory area that holds variables used by MVS data management access methods. It is of interest only to Customer Support.

Data Set

For a mainframe environment, a collection of related records stored together and located by means of a catalog entry.

For the ONC/NFS environment, a contiguous string of characters. New lines are recognized, but the record boundaries do not exist.

For the UNIX environment, referred to as a file.

Data Set Organization

The way in which data is arranged within a data set on a mainframe is called the data set organization. Only sequential, direct, partitioned, and VSAM data set organizations are supported by SNS/NFS. (See **Sequential Data Set**, **Key-Sequenced Data Set**, **Direct Data Set**, **Partitioned Data Set**.)

Datagram

The basic unit of information that is passed across the internet. It consists of one or more data packets (e.g., sets of data).

Defense Advanced Research Projects Agency (DARPA)

A military agency whose network, the ARPANET, was the first to use the TCP/IP protocol over its Wide Area Network (WAN). DARPA has sponsored the development of wide area networks and networking software.

Defense Data Network (DDN)

A network that comprises several Department of Defense networks.

Direct-Access Storage Device (DASD)

A device in which access to data is independent of where data resides on the device.

Direct Data Set (DDS)

A data set type used in a mainframe environment for storing data on a random access device that is accessed using a record address. (See **Sequential Data Set** and **Partitioned Data Set**.)

Directory Services Parameter List (DPL)

A list used to pass information for processing directory service requests such as host name resolution, host address resolution, etc.

Disk Operating System (DOS)

An operating system developed by Microsoft for IBM and compatible personal computers.

Document Composition Facility (DCF)

A text processing program. Its main component is the text formatter, called SCRIPT/VS.

Domain

A collection of machines that is administered as a single entity, having its own unique name.

Domain Name Resolver (DNR)

A facility that provides name resolution services to the ACP task group and to other address spaces via cross-memory services.

Domain Name System (DNS)

(See **Network Directory Services**.)

External Gateway Protocol (EGP)

Protocol used by exterior neighbors to advertise reachability information to other autonomous systems.

Endpoint

(See **Transport Endpoint**.)

Enterprise Print Services (EPS)

Software for MVS TCP/IP that provides printer access across the network. EPS lets MVS users send data to printers attached to other systems on the TCP/IP network and other users on other systems on the TCP/IP network send data to printers controlled by the MVS JES2/JES subsystems.

Entity

Open Systems Interconnection (OSI) terminology for a layer protocol machine. An entity within a layer performs the functions of the layer within a single computer system, accessing the layer entity below and providing services to the layer entity above at local service access points.

Entry-Sequenced Data Set (ESDS)

A type of data set used in a mainframe environment. The format consists of logical records sequenced by the time of their arrival. A particular record is located using the Relative Byte Address (RBA).

Ethernet

A commonly-used, local area network technology originally developed by the Xerox Corporation. This is a form of network that provides real-time communication between machines.

Event Control Block (ECB)

An MVS control block used to communicate between MVS services and application or system modules.

Exit

A mechanism that provides a controlled interface from a server application to a user-provided function. Exits are used in SNS/NFS to interface to site security, archival storage, and accounting software.

Expedited Data

(See **Out-of-Band**.)

Extended Binary-Coded Decimal Interchange Code (EBCDIC)

A set of 256 characters consisting of eight-bit coded characters.

External Control Block (XCB)

A data area that contains the variables used by the NFS server task for the interface to MVS data management for a particular file access.

200801-020080-310100

External Data Representation (XDR)

An encoding convention for data transfer between computers on a network. XDR specifies byte order, record padding, etc. in an architecture-independent manner.

Fastpath

An Intel 977*x* server controller.

FDDI Channel Attachment (FCA)

An implementation for FDDI in 3762.

Fiber Distributed Data Interface (FDDI)

An emerging high-speed networking standard, the underlying medium of which is fiber optics, and the topology is a dual-attached, counter-rotating Token Ring.

File

An ordered collection of data in a UNIX environment stored or processed as a unit. This collection of data is called a data set in a mainframe environment.

File Handle Database (FHDB)

A data set maintained by the NFS server that holds information about workstation mount points across separate executions of the server task.

File Transfer, Access, and Management (FTAM)

The Open Systems Interconnection (OSI) remote file service and protocol.

File Transfer Protocol (FTP)

A protocol (and program) used to transfer files between host machines in a TCP/IP network.

File System

A hierarchical arrangement of directories and files.

File Usage Block (FUB)

An Internal Control Block (ICB) used to contain variables during an individual access (read, write, etc.) on a file within NFS.

GateD

Program that implements dynamic routing protocols such as RIP, HELLO, EGP, and OSPF.

Gateway

A functional unit that interconnects a local data network with another network having different protocols. It enables networks using different protocols to communicate with each other.

Global Storage Block (GSB)

A data area created and used by the NFS user exits. The GSB address is maintained by the NFS server for the duration of its execution.

Global System Options (GSO)

A definition file that defines system options, such as operating and performance parameters, and site exits.

Graphical User Interface (GUI)

A high-level interface that uses windows and menus with graphic symbols instead of typed system commands to provide an interactive environment for a user.

Gratuitous ARP Reply (GARP)

An ARP reply that is issued whenever a driver terminates. This is part Fault Tolerance.

High-level Index

An entry in the system catalog.

IDC Access Method Services (IDCAMS)

(See **Access Method Services**.)

Indexed Data Set

(See **Key-Sequenced Data Set**.)

Information Management System (IMS)

An IBM-licensed product that provides transaction-based processing on MVS.

Infrastucture (IFS)

A software product that provides a multiple thread execution environment for MVS tasks. SNS/TCPaccess uses IFS.

Initial Program Load (IPL)

The process of readying an IBM mainframe system for operation. The IPL loads a series of software programs from disk and initiates system tasks.

Institute of Electrical and Electronic Engineers (IEEE)

An organization that sets standards for computers and communications.

Instruction Length Code (ILC)

On an IBM/370 system, a two-byte memory location that contains the length of the instruction being executed.

Interior Gateway Protocol (IGP)

Protocol used by interior gateways to exchange reachability and routing information.

Interactive Systems Productivity Facility (ISPF)

A TSO utility that lets users edit data sets in full-screen mode, submit jobs, and retrieve output from a 3270 display terminal (or equivalent).

Interface

A shared boundary between two or more entities. This might be a hardware or software component that links two devices or programs together.

Internal Control Block (ICB)

A control block data structure used exclusively by SNS/NFS during its operation. The layout and contents may change without notice and are of interest only to Customer Support.

International Organization for Standards (ISO)

Best known for the seven-layer OSI Reference Model, a standard model of architecture that constitutes the framework for the development of standard protocols. Previously known as the International Standards Organization. (See **Open Systems Interconnection**.)

International Standards Organization (ISO)

(See **International Organization for Standards**)

International Telegraph and Telephone Consultative Committee (CCITT)

An international advisory body whose charter is to develop international standards for telegraph and telephone companies.

Inter-network Protocol (IP)

The TCP/IP layer between the higher level host-to-host protocol and the local network protocols. IP uses local area network protocols to carry packets, in the form of datagrams, to the next gateway, router, or destination host.

Internet

A global web of interconnected university, business, military, and science computer networks. This network of networks consists of tens-of-thousands of computer systems connected to thousands of interconnected networks.

Internet Control Message Protocol (ICMP)

A standard protocol that is an integral part of IP but uses IP as a higher-level protocol. It lets a gateway or destination host communicate with the source host to report errors on any IP datagram with the exception of ICMP messages.

I/O Configuration Program

A program that defines to a system all available I/O devices and channel paths.

200801-020080-310100

Job Control Language (JCL)

A series of instructions to the MVS operating system describing which programs to run and which data sets to associate with those programs.

Job Entry Subsystem (JES)

A system facility for spooling, job queueing, and managing I/O.

Kernel

A master program set of UNIX software that manages all the physical resources of the computer, including file system management, virtual memory, reading and writing files to disks and tapes, scheduling of processes, and printing and communicating over a network.

Key-Sequenced Data Set (KSDS)

A type of data set used in a mainframe environment for storing data on a random access device. The format consists of an index followed by one or more logical records. This is also known as indexed data set.

Linear Data Set (LDS)

A named linear string of data stored in such a way that it can be retrieved or updated in 4096-byte units. A linear data set object is essentially a VSAM entry-sequenced data set that is processed as an unstructured byte stream.

Link State Advertisement (LSA)

A family of OSPF packets that contain information on routes and routers.

Little-endian

A format for storage or transmission of binary data in which the least significant byte (or bit) comes first. (See **Big-endian**.)

Local Area Network (LAN)

A data network located on the user's premises in which serial transmission is used for direct data communication among data stations.

Logical Unit (LU)

In SNA, a port through which an end user accesses the SNA network and the functions provided by system services control points. An LU can support at least two sessions and may be capable of supporting many sessions with other LUs. The type of services provided by the LU to the end user differs for each type of LU. LU types 2 and 3 support communication between processes and terminals. LU 6.2 supports general communication between programs in a distributed processing environment. (See **Advanced Peer-to-Peer Communication**.)

Logon ID (LID)

A record that contains the logon identification of the user. This is an ACF2 term.

Mainframe

A computer running the MVS operating system, including, but not limited to, machines from IBM, Amdahl, Fujitsu, and NAS.

Member

A term for an independent group of sequentially-organized records in a partitioned data set used on a mainframe. Message Handling System (MHS)

The system of message user agents, message transfer agents, message stores, and access units that together provide Open Systems Interconnection (OSI) electronic mail.

Mount

A process to access a directory from a local disk attached to the machine making the mount request (UNIX mount) or remote disk on a network (NFS mount).

An operation that associates a group of files on a server with a directory (mount point) on a client to provide transparent access to the files through that directory. The files must be in a hierarchical arrangement.

Mount Point

A place established in a workstation or server local directory that is used during the transparent accessing of a remote file.

020080-310100

Multiple Virtual Storage (MVS)

An IBM-licensed multipurpose operating system for the System/370 and System/390 processors. MVS handles three major functional areas: job management (e.g., command processing and job processing), supervisor management (e.g., resource management), and data management (e.g., file and input/output operations). MVS supports such characteristics as a large number of concurrent users, teleprocessing, and efficient storage and retrieval from large databases.

Network Data Set Header (NDH)

An NJE control record that generally precedes a unit of SYSOUT data. It may also appear in the middle of SYSIN data to indicate a change in the format of the *SYSIN* data.

Network Directory Services (NDS)

A distributed database system that provides dynamic information retrieval for host-specific address and name information.

Network File System (NFS)

A distributed component of ONC developed by Sun Microsystems that lets a set of computers cooperatively access each other's files transparently. Once accessed the filesystem appears to reside on the local host.

Network Information Center (NIC)

A central administration facility for the Internet. The NIC supervises network names and network addresses, and provides several information services.

Network Job Entry (NJE)

A protocol that enables jobs to be submitted and output returned remotely.

Network Service Access Point (NSAP)

The point at which the OSI network service is made available to a transport entity.

Network Virtual Terminal (NVT)

A set of rules defining a very simple virtual terminal interaction. The NVT is used at the start of a TELNET session, but a more complex type of terminal interaction can be negotiated.

Node Initialization Block (NIB)

In VTAM, a control block associated with a particular node or session that contains information used by the application program to identify the node or session and to indicate how communication requests on a session are to be handled by VTAM.

Non-Broadcast Multi-Access (NBMA)

A non-point-to-point interface that is not broadcast-capable.

Open Network Computing (ONC)

A set of software that supports heterogeneous distributed computing.

Open Shortest Path First (OSPF)

Routing protocol for interior gateways that allows dynamic routing choices to be implemented.

Open System

A system that is internally independent but provides common external services via standardized protocols publicly defined for each layer. (See **Open Systems Interconnection**.)Open Systems Interconnection (OSI)

An international standardization program to facilitate communications among computers from different manufacturers. (See **International Standards Organization**.)

Operating System/2 (OS/2)

A multi-tasking operating system for IBM and compatible personal computers, developed by IBM and Microsoft.

Out-of-Band (OOB)

A mechanism in TCP used to bypass flow control and reach the server immediately. Also known as urgent data or expedited data.

200801-020080-310100

Partitioned Data Set (PDS)

A direct access storage data set that is divided into partitions, called members, each of which can contain a program, part of a program, or data. The PDS has a directory that points to the locations of the various members stored in this data set. This directory allows the members to be independently accessed. Partitioned data sets are often used to store libraries of programs and macro instructions. (See **Sequential Data Set**, **Direct Data Set**).

Partitioned Data Set Extended (PDSE)

A storage management subsystem-managed, page-formatted, data set that is divided into partitions called members, each of which can contain non-load module data.

Pipeline

A connection between the standard output of one process and the standard input of a second process.

Portable Operating System Environment for Computer Environments (POSIX)

A family of operating system standards developed by IEEE that defines the services that must be provided to be "POSIX compliant".

Portmapper

A program that maps client programs to the port numbers of server programs. The portmapper is used with Remote Procedure Call (RPC) programs.

Primary Logical Unit (PLU)

In SNA, the Logical Unit (LU) that contains the primary half-session for a particular LU-LU session. Each session must have a PLU and Secondary Logical Unit (SLU). The PLU is the unit responsible for the bind and is the controlling LU for the session. A particular LU may contain both primary and secondary half-sessions for different active LU-LU sessions. (See **Secondary Logical Unit (SLU)**.)

Primitive

A pair of functions that provide an unreliable datagram packet transport between two systems.

Process Control Entry (PCE)

The control block that represents an EPS internal process (i.e., an independent thread of execution).

Procedure Correlation Identifier (PCID)

In SNA, a value used by a control point to correlate requests and replies.

Process

A program in operation. For example, a daemon is a system process that is always running on the system. A process is usually logical, not physical.

Program Request Block (PRB)

An internal MVS control block that identifies a user program to the system. This control block is used by MVS task management to dispatch the executable program.

Program Status Word (PSW)

In the 370 architecture, an eight-byte memory location that includes the instruction counter and control flags of a running program. A PSW is part of the state of a running process.

Protocol

A set of rules for communicating between diverse systems. The rules are mutually understood and followed by the different systems or processes. The protocol specifies actions that can be taken by a station when it receives a transmission or detects an error condition.

Protocol Data Unit (PDU)

A generic term for the protocol unit (e.g., a header and data) used at any layer. Referred to as Transport Protocol Data Unit (TPDU) when used within the transport layer.

Prototype

A model suitable for use as a model for developing software or systems.

Provider

(See **Transport Provider**.)

20-020080-310100

Relative Byte Address (RBA)

The displacement of a data item from the beginning of the data set, independent of the manner in which the data set is stored.

Relative Record Data Set (RRDS)

A type of data set used in the mainframe environment. It must be on a direct access volume and the format consists of one logical record in a fixed-length slot. Each slot has a unique relative record number. Data is placed in a specific slot, based on a user-supplied relative record number.

Remote Procedure Call (RPC)

A facility that a client uses to request the execution of a procedure call from a server. This facility includes a library of procedures and an external data representation.

Request Parameter List (RPL)

A data structure used to request a function of a VSAM data set or VTAM Logical Unit (LU). The contents of an RPL include addresses of data, status codes, and internal control parameters.

Request/Response Unit (RU)

In SNA, a message unit that requests a function or responds to a function. The RU contains flags describing the remaining portion of the message.

Resource Access Control Facility (RACF)

An IBM-licensed program that provides for access control by identifying and by verifying the users to the system, authorizing access to protected resources, logging the detected unauthorized attempts to enter the system, and logging the detected accesses to protected resources.

Resource Definition Online (RDO)

A facility that allows for dynamic definition of CICS resources.

Root

A privileged user name, typically in a UNIX environment, that gives the user the ability to modify restricted system files. (See **Superuser**).

ROSCOE

A user interface for MVS providing an alternative to TSO.

Routing Information Protocol (RIP)

Vector-distance routing protocol.

Secondary Logical Unit (SLU)

In SNA, the Logical Unit (LU) that contains the secondary half-session for a particular LU-LU session. An LU may contain secondary and primary half-sessions for different active LU-LU sessions. (See **Primary Logical Unit (PLU)**.)

Sequential Data Set

A type of data set used in the mainframe environment. It must be on a direct access volume with its records stored and retrieved according to their physical order within the data set. (See **Direct Data Set**, **Partitioned Data Set**).

Server

A machine providing resources to the network. It provides a network service, such as disk storage and file transfer, or contains a program that provides such a service.

Service Access Point (SAP)

The point at which the services of an OSI layer are made available to the next higher layer.

Service Data Unit (SDU)

The unit of data transferred across the interface between a user and provides, and subsequently between two peer users.

Set Buffer Address (SBA)

A control sequence in a 3270 data stream that specifies a new display address to associate with the data that follows.

Sharing

A term used in a computing environment to refer to using a file on a remote system. This is done by mounting the remote file system, then reading from or writing to files in that remote system.

200801-020080-310100

Sim3278/TCPIP Host Program Interface (SIMPCS)

Software manufactured by Simware that is used to enable VT-class terminals to invoke User telnet services in SNS/TCPACCESS.

Simple Mail Transfer Protocol (SMTP)

A protocol that defines how mail is sent in a TCP/IP environment.

SNA Character String (SCS)

In SNA, a character string composed of EBCDIC controls, optionally intermixed with end-user data, that is carried within a request/response unit. Controls EBCDIC control codes in a printer data stream to format printed pages, set modes of device operation, define unique data, or communicate between a user and an application program.

SNS/TCPaccess

A communication subsystem for the internet protocols. It runs on an IBM 370-style mainframe under the MVS operating system.

Socket

An end point for communication that can be named and addressed in a network. The socket interface is one of several Application Program Interfaces (APIs) to the communication protocols.

Software Network Solution (SNS)

The Interlink family of software products that enables communication between networks and mainframes.

Spool Display and Search Facility (SDSF)

An IBM-licensed product that permits time-sharing users to browse and administer the JES2 spool data sets.

Started Task

An MVS system-related job that is initiated by operators from their consoles.

Started Task Control (STC)

A task started from the operator console of a system.

Storage Management System (SMS)

An IBM program product that provides hierarchic management of mainframe disk storage resources.

Subnet

A networking scheme that divides a single logical network into smaller physical networks to simplify routing.

Superuser

A user with special privileges granted if the correct password is supplied when logging in as root or using the `su` command.(See **Root**.)

Supervisor Call (SVC)

The instruction used by the mainframe to generate a software interrupt. Control is then transferred to a routine that handles the interrupt processing. SVC 99 is used to dynamically allocate and deallocate data sets.

System Authorization Facility (SAF)

A system that gets control in response to a request from a resource manager. SAF conditionally directs control to RACF or an installation-supplied processing routine if present, or both.

System Catalog

The highest level catalog on a mainframe and must exist so that the operating system can find data files. It is a VSAM data set and can contain pointers to VSAM data sets, VSAM user catalogs, OS data sets, and OS user catalogs.

System Diagnostic Work Area (SDWA)

An MVS control block that is passed to an abnormal termination recovery routine by the operating system. The SDWA contains the system state at the time of the failure.

System Management Facility (SMF)

A function used on the mainframe to log accounting information, including processor time, data transfer statistics, and user information.

System Modification Program (SMP)

An operating system component that facilitates the process of installing and servicing an MVS system. (See **System Modification Program/ Extended**.)

System Modification Program/Extended (SMP/E)

An IBM system programming utility that manages and installs software components into the MVS operating system environment. (See **System Modification Program**.)

System Network Architecture (SNA)

An IBM-licensed network architecture that defines the rules for synchronous communication between two devices in a network. SNA describes the logical structure, formats, protocols, and operational sequences for transmitting information units through networks, and for controlling the configuration and operation of networks.

System Services Control Point (SSCP)

In SNA, a central location point within an SNA network for managing the configuration, coordinating network operation and problem determination requests, and providing directory support and other session services for end users of the network.

TELNET

The virtual terminal protocol in the Internet suite of protocols. Lets a user of one host log into a remote host and interact as a normal terminal user of that host.

Time Sharing Option (TSO)

An MVS operating system feature that provides interactive execution of programs and time sharing from remote terminals.

Token

In a local area network, the symbol of authority that is passed from one data station to another. The token indicates the data station that currently has transmission control on the network.

TOP SECRET

(See **CA-TOP SECRET**.)

Transmission Control Protocol (TCP)

The TCP/IP layer that provides reliable, process-to-process data stream delivery between nodes in interconnected computer networks. TCP assumes that IP is the underlying protocol.

Transmission Control Protocol/Internet Protocol (TCP/IP)

A suite of protocols designed to allow communication between networks regardless of the technologies implemented in each network. This connectivity includes the ability to transfer files, send mail, and log on to a remote host in a network of heterogeneous systems. TCP/IP forms the base technology for a large internet that connects most major research institutions, including university, corporate, and government labs.

Transport

A method by which data is moved across a network. The Transport Layer Interface (TLI) lets programming interfaces be built, which ensure that the transport mechanism transmits data to the network reliably.

Transport Connection

A connection between two transport users.

Transport Connection Endpoint (TCEP)

The termination of a connection between two transport users within a TSAP.

Transport Endpoint

(See **Transport Connection Endpoint (TCEP)**.)

Transport Layer

The Open Systems Interconnection (OSI) layer that is responsible for reliable end-to-end data transfer between TCP systems. (See **Transmission Control Protocol**.)

200801-020080-310100

Transport Layer Interface (TLI)

A programmable interface that lets applications be built as independent of the networking protocols below them, and provides reliable network transmission.

Transport Protocol Class 0 - 4 (TP0-4)

The five Open Systems Interconnection (OSI) transport protocols. TP0 (Simple Class) is the simplest and is useful only on top of an X.25 network (or other network that does not lose or damage data). TP4 (Error Detection and Recovery Class) is the most powerful layer and is useful on top of any type of network.

Transport Protocol Data Unit (TPDU)

(See **Protocol Data Unit (PDU)**.)

Transport Provider

The interface that allows application programs access to network transport services.

Transport Provider Address Space (TPAS)

The address space containing the SNS/TCPaccess API subsystem and the transport service provider that processes a service request.

Transport Service Access Point (TSAP)

A communication path between a transport user and a specific transport provider.

Transport Service Data Unit (TSDU)

See **Service Data Unit (TSDU)**.)

Transport Service Parameter List (TPL)

The primary structure for requesting API services and exchanging information between the application program and the transport services.

Transport Services Information Block (TIB)

A data area that specifies parameter values of the API TCP and UDP transport services.

Transport User Address Space (TUAS)

The address space of an application program initiating a service request.

Unbinding

A function that removes the association of a variable with the address or point to which it is bound. (See **Binding**.)

UNIX Filesystem (UFS)

An NFS feature that enables storage of UNIX-style files (byte streams) in MVS storage volumes. UFS files are not translated or reformatted.

Urgent Data

(See **Out-of-Band**.)

User

A person or program function that uses the services provided by a server. A host can be a user and a server at the same time.

User Datagram Protocol (UDP)

A datagram-level protocol built directly on the IP layer. UDP is used for application-to-application programs between TCP/IP hosts. NFS uses UDP as its low-level transport protocol.

User ID (UID)

A token that identifies a user to an operating system. In the UNIX environment, the UID is a numeric value. MVS generally refers to the user identification value as a username.

User Storage Block (USB)

A data area created and used by the NFS user exits. The USB address is maintained by the NFS server for the duration of a user session.

Virtual Machine (VM)

An IBM-licensed operating system that manages the resources of a single computer so that multiple computing systems appear to exist. Each virtual machine is the functional equivalent of a physical (i.e., "real") machine.

020080-310100

Virtual Memory System (VMS)

An interactive operating system that lets you conduct a dialog of command and response with the system.

Virtual Printer (VPRT)

A software simulation of a physical printer that responds to the same print formats as an actual printer. In EPS, the virtual printer subtask is called VPRT.

Virtual Storage Access Method (VSAM)

An access method for direct or sequential processing of fixed and variable-length records on direct access devices. The records in a VSAM data set or file can be organized in logical sequence by a key field (key sequence), in the physical sequence in which they are written on the data set or file (entry-sequence), or by relative-record number.

Virtual Telecommunications Access Method (VTAM)

An IBM-licensed program that controls communication between nodes and application programs, as well as the flow of data in an SNA network. It provides single-domain, multiple-domain, and interconnected network capability, and lets application programs communicate with other programs and devices in a network.

Virtual Terminal (VT)

An intelligent device, such as a personal computer, that appears to be a terminal to the host computer to which it is connected.

Volume Table of Contents (VTOC)

A table on a direct access volume that describes each data set on the volume.

Wide Area Network (WAN)

A network that consists of machines connected over a wide geographic area. The ARPANET is an example of a wide area network.

Workstation

Any machine or computer on a network that implements communication between enhanced connectivity programs. These communicating programs reside on both the workstation and host system.

Write to Operator with Reply (WTOR)

An MVS system services function that presents a message to the system operator and allows a reply.

200801-020080-310100

Acronyms

ABEND

Abnormal End of Task

ACB

Access Method Control Block

ACF2

Access Control Facility 2

ACEE

Accessor Environment Element

ACID

Access Control ID

ADB

Attribute Database

AFP

Advanced Function Printing

AMDCB

Access Module Control Block

AMS

Access Method Services

ANSI

American National Standards Institute

APCB

Application Program Control Block

APF

Authorized Program Facility

API

Application Program Interface

APPC

Advanced Peer-to-Peer Communication

APPN

Advanced Peer-to-Peer Networking

ARP

Address Resolution Protocol

ASCII

American Standard Code for Information Interchange (from ANSI)

BDT

Bulk Data Transfer

BLDL

Build a Directory Entry List

BMS

Basic Mapping Support

BSD

Berkeley Software Distribution

CCITT

International Telegraph and Telephone Consultative Committee

CEP

Connection Endpoint

CICS

Customer Information Control System

CID

Connection ID

CIF

Caltech Intermediate Form

CLNP

Connectionless Network Protocol

20 020080-310100

CLNS
Connectionless Network Service

CLTP
Connectionless Transport Protocol

CLTS
Connectionless Transport Service

COTS
Connection-Oriented Transport Service

CPT
CICS Programmer's Toolkit

DARPA
Defense Advanced Research Projects Agency

DASD
Direct-Access Storage Device

DCB
Data Control Block

DCF
Document Composition Facility

DDN
Defense Data Network

DDS
Direct Data Set

DNR
Domain Name Resolver

DNS
Domain Name System

DOS
Disk Operating System

DPL
Directory Services Parameter List

EBCDIC
Extended Binary Coded Decimal Interchange Characters (from IBM)

ECB
Event Control Block

EGP
External Gateway Protocol

EPS
Enterprise Print Services

ESDS
Entry-Sequenced Data Set

FCA
FDDI Channel Attachment

FDDI
Fiber Distributed Data Interface

FHDB
File Handle Database

FTAM
File Transfer, Access, and Management

FTP
File Transfer Protocol

FUB
File Usage Block

GARP
Gratuitous ARP Reply

GSB
Global Storage Block

GSO
Global System Options

200801-020080-310100

GUI

Graphical User Interface

ICB

Internal Control Block

ICMP

Internet Control Message Protocol

IDCAMS

IDC Access Method Services

IEEE

Institute of Electrical and Electronic Engineers

IFS

Infrastructure

ILC

Instruction Length Code

IMS

Information Management System

IGP

Internal Gateway Protocol

IOPC

I/O configuration Program

IP

Inter-network Protocol

IPL

Initial Program Load

ISO

International Organization for Standards (previously known as International Standards Organization)

ISPF

Interactive System Productivity Facility

JCL

Job Control Language

JES

Job Entry Subsystem

KSDS

Key-Sequenced Data Set

LAN

Local Area Network

LDS

Linear Data Set

LID

Logon ID

LSA

Link State Advertisement

LU

Logical Unit

MHS

Message Handling System

MVS

Multiple Virtual Storage

NBMA

Non-Broadcast Multi-Access

NDH

Network Data Set Header

NDS

Network Directory Services

NFS

Network File System

NIB

Node Initialization Block

NIC

Network Information Center

NJE

Network Job Entry

NVT

Network Virtual Terminal

ONC

Open Network Computing

OOB

Out-of-Band

OS/2

Operating System/2

OSI

Open Systems Interconnection

OSPF

Open Shortest Path First

PCE

Process Control Entry

PCID

Procedure Correlation Identifier

PDS

Partitioned Data Set

PDSE

Partitioned Data Set Extended

PDU

Protocol Data Unit

PLU

Primary Logical Unit

POSIX

Portable Operating System Environment for Computer Environments

PRB

Program Request Block

PSW

Program Status Word

RACF

Resource Access Control Facility

RBA

Relative Byte Address

RDO

Resource Definition Online

RIP

Routing Information Protocol

RPC

Remote Procedure Call

RPL

Request Parameter List

RRDS

Relative Record Data Set

RU

Request/Response Unit

SAF

System Authorization Facility

SAP

Service Access Point

SBA

Set Buffer Address

200801-020080-310100

SCS

SNA Character Set

SDSF

Spool Display and Search Facility

SDU

Service Data Unit

SDWA

System Diagnostic Work Area

SLU

Secondary Logical Unit

SMF

System Management Facility

SMP

System Modification Program

SMP/E

System Modification Program/Extended

SMS

Storage Management System

SMTP

Simple Mail Transfer Protocol

SNA

System Network Architecture

SNS

Software Network Solution

SPF

Shortest Path First

SSCP

System Services Control Point

STC

Started Task Control

SVC

Supervisor Call

TCEP

Transport Connection Endpoint

TCP

Transmission Control Protocol

TCP/IP

Transmission Control Protocol/Internet Protocol

TIB

Transport Service Information Block

TLI

Transport Layer Interface

TP0 - TP4

Transport Protocol Class 0 through Transport Protocol Class 4

TPL

Transport Service Parameter List

TPAS

Transport Provider Address Space

TPDU

Transport Protocol Data Unit

TSAP

Transport Service Access Point

TSDU

Transport Service Data Unit

TSO

Time Sharing Option

TUAS

Transport User Address Space

020080-310100

UDP

User Datagram Protocol

UFS

UNIX File System

UID

User ID

USB

User Storage Block

VM

Virtual Machine

VMS

Virtual Memory System

VPRT

Virtual Printer

VSAM

Virtual Storage Access Method

VT

Virtual Terminal

VTAM

Virtual Telecommunications Access Method

VTOC

Volume Table of Contents

WAN

Wide Area Network

WTOR

Write To Operator with Reply

XCB

External Control Block

XDR

External Data Representation

200801-020080-310100

Index

200801-024040-200100

200 024040-200100

D

E

F

G

200801-024040-200100

L

200 024040-200100

M

P

R

200801-024040-200100

S

200 24040-200100

T

200801-024040-200100

U

W

200 024040-200100

Reader Comment Form

Please let us know if you find any errors in this document. We would also like to hear your comments, questions, or suggestions.

You can write to us at this address:

Interlink Computer Sciences, Inc.
Technical Publications Department
47370 Fremont Boulevard
Fremont, California 94538
U.S.A.

You can email us at this address:

techpubs@interlink.com

You can fax us at this number:

(510) 659-6381

If you have corrections or comments, please include this information:

- Document name or number (found on the front cover or the inner margin of each page)
- Page number

We appreciate your corrections and comments.

INTERLINK
Computer Sciences

Interlink Computer Sciences, Inc.
Corporate Headquarters
47370 Fremont Blvd.
Fremont, CA 94538
(510) 657-9800
(800) 422-3711
fax (510) 659-6381
email: info@interlink.com
web: http://www.interlink.com

For other U.S. sales offices
call (800) 422-3711

Canada
(403) 231-9800
fax (403) 266-6767

France
+33-1-45-079494
fax +33-1-45-079851

Germany
+49-2203-911853
fax +49-2203-911859

Spain
+34-1-352-62-65
fax +34-1-352-48-76

Switzerland
+41-26-460-85-50
fax +41-26-460-85-58

United Kingdom
+44-1923-77-6000
fax +44-1923-77-1100

200801-024040-200100