Unicenter® SOLVE:CPT™

Assembler Programmers Guide

r6.1 SP2

This documentation and any related computer software help programs (hereinafter referred to as the "Documentation") is for the end user's informational purposes only and is subject to change or withdrawal by CA at any time.

This Documentation may not be copied, transferred, reproduced, disclosed, modified or duplicated, in whole or in part, without the prior written consent of CA. This Documentation is confidential and proprietary information of CA and protected by the copyright laws of the United States and international treaties.

Notwithstanding the foregoing, licensed users may print a reasonable number of copies of the documentation for their own internal use, and may make one copy of the related software as reasonably required for back-up and disaster recovery purposes, provided that all CA copyright notices and legends are affixed to each reproduced copy. Only authorized employees, consultants, or agents of the user who are bound by the provisions of the license for the product are permitted to have access to such copies.

The right to print copies of the documentation and to make a copy of the related software is limited to the period during which the applicable license for the Product remains in full force and effect. Should the license terminate for any reason, it shall be the user's responsibility to certify in writing to CA that all copies and partial copies of the Documentation have been returned to CA or destroyed.

EXCEPT AS OTHERWISE STATED IN THE APPLICABLE LICENSE AGREEMENT, TO THE EXTENT PERMITTED BY APPLICABLE LAW, CA PROVIDES THIS DOCUMENTATION "AS IS" WITHOUT WARRANTY OF ANY KIND, INCLUDING WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT. IN NO EVENT WILL CA BE LIABLE TO THE END USER OR ANY THIRD PARTY FOR ANY LOSS OR DAMAGE, DIRECT OR INDIRECT, FROM THE USE OF THIS DOCUMENTATION, INCLUDING WITHOUT LIMITATION, LOST PROFITS, BUSINESS INTERRUPTION, GOODWILL, OR LOST DATA, EVEN IF CA IS EXPRESSLY ADVISED OF SUCH LOSS OR DAMAGE.

The use of any product referenced in the Documentation is governed by the end user's applicable license agreement.

The manufacturer of this Documentation is CA.

Provided with "Restricted Rights." Use, duplication or disclosure by the United States Government is subject to the restrictions set forth in FAR Sections 12.212, 52.227-14, and 52.227-19(c)(1) - (2) and DFARS Section 252.227-7014(b)(3), as applicable, or their successors.

All trademarks, trade names, service marks, and logos referenced herein belong to their respective companies.

Copyright © 2006 CA. All rights reserved.

Contents

Chapter 1: Unicenter SOLVE:CPT Tools

The Unicenter SOLVE:CPT Administrative Interface . . . 1-2
Unicenter SOLVE:CPT API Services . . . 1-2
The Unicenter SOLVE:CPT FTP Client Callable Service . . . 1-2
Automated Transactions . . . 1-3
Tools Customization . . . 1-3
The LISTEN Tool . . . 1-4
Diagnostics . . . 1-5
MRO Feature . . . 1-5
The RECEIVE Tool . . . 1-5
Customization . . . 1-6
Reliability Factors . . . 1-6
The SEND Tool . . . 1-8
Customization . . . 1-8
Reliability Factors . . . 1-9
The SELECT Tool . . . 1-11

Chapter 2: Unicenter SOLVE:CPT API Services

The Unicenter SOLVE:CPT Interface . . . 2-2
Unicenter SOLVE:CPT Task-Related User Exit Interface (TRUE) . . . 2-3
Application Programming Concepts . . . 2-3
TCP Connection Management . . . 2-5
LISTEN . . . 2-6
CONNECT . . . 2-6
TCP Data Transfer . . . 2-7
SEND . . . 2-8
RECEIVE . . . 2-8
UDP Data Transfer and Endpoint Creation . . . 2-9
SENDTO . . . 2-10
RCVFROM . . . 2-10

Connection and Endpoint Release . . . 2-11
CLOSE . . . 2-11
Data Translation . . . 2-12
TRANSLATE . . . 2-12
Facility Management . . . 2-13
GIVE . . . 2-14
TAKE . . . 2-15
Unicenter SOLVE:CPT FTP Client Service . . . 2-16
Unicenter SOLVE:CPT FTP Client Service Overview . . . 2-17
Security Program . . . 2-19
Security Program . . . 2-20
The Security Communications Block . . . 2-21
Security Communications Block . . . 2-22
Sample Unicenter SOLVE:CPT API Pseudo Code . . . 2-23
Client Application Example . . . 2-23
Server Application Example 1 . . . 2-25
Server Application Example 2 . . . 2-27
Server Application Example 3 . . . 2-29
Server Application Example 4 . . . 2-31
Unicenter SOLVE:CPT API Sample Programs . . . 2-32
Client 1 Sample Program . . . 2-35
TCP Client 2 Sample Program . . . 2-36
TCP Server 1 Sample Program . . . 2-36
TCP Server 2 Sample Program . . . 2-37
Server 3 Sample Program . . . 2-37
Server 4 Sample Program . . . 2-37
Server 5 Sample Program . . . 2-38
UDP Client Sample Program . . . 2-38
UDP Server Sample Program . . . 2-38
Using CA-InterTest® with Unicenter SOLVE:CPT Applications . . . 2-39
Compiling and Linking a CPT API Application . . . 2-40

Chapter 3: CLOSE Service

Call Syntax . . . 3-2
Recommended ACL Parameters . . . 3-2
Usage Examples . . . 3-3
Parameter Values Returned in the ACL . . . 3-5
Assembler DSECTS . . . 3-5
Sample Programs . . . 3-6
Completion Information . . . 3-7

Return Codes . . . 3-8
Usage Notes . . . 3-9
Complete Parameter List . . . 3-11

Chapter 4: CONNECT Service

Call Syntax . . . 4-2
Recommended ACM Parameters . . . 4-2
Usage Example . . . 4-3
Parameter Values Returned in the ACM . . . 4-4
Assembler DSECTs . . . 4-4
Sample Programs . . . 4-5
Completion Information . . . 4-5
Return Codes . . . 4-6
Usage Notes . . . 4-8
Complete Parameter List . . . 4-10

Chapter 5: FTP Client Service

Call Syntax . . . 5-2
Recommended AFT Parameters . . . 5-2
Usage Example . . . 5-3
    Sample Program T09PAFT3 . . . 5-5
    Parameter Values Returned in the AFT . . . 5-7
Assembler DSECTs . . . 5-7
Sample Programs . . . 5-8
Completion Information . . . 5-8
Return Codes . . . 5-9
Module Descriptions . . . 5-11
    T09TCFCM . . . 5-11
    T09TCFDM . . . 5-12
    T09TCFRM . . . 5-12
Usage Notes . . . 5-13
Complete Parameter List . . . 5-14

Chapter 6: GIVE Service

Call Syntax . . . 6-2
Recommended AFM Parameters . . . 6-2
Usage Example . . . 6-3
Parameter Values Returned in the AFM . . . 6-4
Assembler DSECTs . . . 6-4
Sample Programs . . . 6-4
Completion Information . . . 6-5
Return Codes . . . 6-5
Usage Notes . . . 6-6
Complete Parameter List . . . 6-7

Chapter 7: LISTEN Service

Call Syntax . . . 7-2
Recommended ACM Parameters . . . 7-3
Usage Examples . . . 7-3
    Recommended Server . . . 7-4
    Standard Multithreaded Server . . . 7-4
    Multithreaded Server—Special Start Transaction Needs . . . 7-5
    Single-Threaded Server . . . 7-7
    Sample Daughter Task Taking Ownership of a Session . . . 7-8
Parameter Values Returned in the ACM . . . 7-9
Assembler DSECTs . . . 7-9
Sample Programs . . . 7-10
Completion Information . . . 7-11
    Completion Information When the ACMTRNID Field Is Set in the CPT-ACM . . . 7-11
    Completion Information When the ACMTRNID Field Is Unspecified in the CPT-ACM . . . 7-12
Return Codes . . . 7-13
Usage Notes . . . 7-15
Network Considerations . . . 7-17
Complete Parameter List . . . 7-18
Client-Data Listener Option . . . 7-24
    Client-Data Option Data Structure . . . 7-26
    Examples . . . 7-26
    Invoking the Listener with Translation from an Assembler CPT . . . 7-27
    Example of a Assembler Program, Client-Data Listener . . . 7-28

Chapter 8: RCVFROM Service

Call Syntax 8-3
Recommended ADT Parameters 8-3
Usage Example 8-4
Parameter Values Returned in the ADT 8-5
Assembler DSECTs 8-6
Sample Programs 8-6
Network Considerations 8-7
Return Codes 8-8
Complete Parameter List 8-10

Chapter 9: RECEIVE Service

Call Syntax 9-2
Receive Methodology Options 9-2
    Terminology and Receive Concepts Used in the Definitions 9-3
    Introduction to Receive Methodology Options 9-6
    Non-Blocking Fixed Length RECEIVE 9-8
    Non-Blocking Variable Length RECEIVE 9-12
    Recommended ADT Parameters 9-12
    Non-Blocking LL RECEIVE 9-16
    Non-Blocking Separator Character RECEIVE 9-20
    Blocking Fixed Length RECEIVE 9-24
    Blocking LL RECEIVE 9-26
    Blocking Separator Character RECEIVE 9-28
    Non-Blocking RECEIVE 9-30
    Blocking RECEIVE 9-32
Parameter Values Returned in the ADT 9-34
Assembler DSECTs 9-34
Sample Programs 9-35
Completion Information 9-36
Return Codes 9-37
Usage Notes 9-39
Complete Parameter List 9-41

Chapter 10: SEND Service

Call Syntax . . . 10-2
Recommended ADT Parameters . . . 10-3
LL SEND Option . . . 10-4
Separator Character SEND Option . . . 10-4
Usage Examples . . . 10-5
Data SEND Example . . . 10-6
LL SEND Example . . . 10-7
Separator Character SEND Example . . . 10-8
Parameter Values Returned in the ADT . . . 10-9
Assembler DSECTs . . . 10-9
Sample Programs . . . 10-10
Completion Information . . . 10-10
Return Codes . . . 10-12
Usage Notes . . . 10-13
Complete Parameter List . . . 10-15

Chapter 11: SENDTO Service

Call Syntax . . . 11-3
Recommended ADT Parameters . . . 11-3
Usage Example . . . 11-4
Parameter Values Returned in the ADT . . . 11-5
Assembler DSECTs . . . 11-5
Sample Programs . . . 11-6
Network Considerations . . . 11-6
Return Codes . . . 11-7
Complete Parameter List . . . 11-9

Chapter 12: TAKE Service

Call Syntax . . . 12-2
Recommended AFM Parameters . . . 12-2
Usage Example . . . 12-3
Parameter Values Returned in the AFM . . . 12-4
Assembler DSECTs . . . 12-4
Sample Programs . . . 12-5
Completion Information . . . 12-5
Return Codes . . . 12-6
Usage Notes . . . 12-7

Complete Parameter List 12-9

Chapter 13: TRANSLATE Service

Call Syntax 13-2
Recommended AXL Parameters 13-2
Usage Example 13-3
Inbound Translation Example 13-3
Outbound Translation Example 13-4
Custom Translation Table Usage Notes 13-5
Parameter Values Returned in the AXL 13-6
Assembler DSECTs 13-6
Sample Programs 13-7
Completion Information 13-7
Return Codes 13-8
Usage Notes 13-9
Complete Parameter List 13-10

Appendix A: Return Codes

Return Codes A-1
Diagnostic Code Field A-4
Assembler MACRO T09DRTCD A-4

Appendix B: Control Block Layouts

ACL: Argument for CLose Used by the CLOSE API Service B-2
Offset Table B-2
Alphabetized Field Name Cross-Reference Table B-3
Sample DSECT Member T09DACL B-4
ACM: Argument for Connection Management Used by the CONNECT and LISTEN API ServicesB-5
Offsets B-5
Alphabetized Field Name Cross-Reference Table B-7
Sample DSECT Member T09DACM B-8
ADT: Argument for Data Transfer Used by the RECEIVE, SEND, RECVFROM, and SENDTO Services B-10
Offsets B-10
Alphabetized Cross-Reference Table B-11
Sample DSECT Member T09DADT B-12

AFM: Argument for Facility Management Used by the GIVE and TAKE Services B-14
Offsets B-14
Alphabetized Cross-Reference Table B-15
Sample DSECT Member T09DAFM B-16
AFT: Argument for File Transfer Used by the FTP Client Service Call B-17
Offsets B-17
Alphabetized Cross-Reference Table B-17
Sample DSECT Member T09DAFT B-18
AXL: Argument for Data Translation Used by the Translate API Service B-20
Offsets B-20
Alphabetized Cross-Reference Table B-22
Sample DSECT Member T09DAXL B-23
Client Data Listener Transaction Start B-24
Sample DSECT Member T09DCSKL B-24
Connection Time Security Program Control Block B-25
Offsets B-25
Sample DSECT Member T09DSCTY B-26
Parameter List Passed to T09MTRAN Initiated Transactions B-27
T09DLST DSECT Sample B-27
Field Descriptions B-28
LCA0000 and CFG0000 Control Blocks B-28
LCA0000 Control Block Listing B-29
CFG0000 Control Block Listing B-29

Appendix C: T09MTRAN Programming Notes

Parameter List Passed to T09MTRAN Initiated Transactions C-2
Basic Layout Usage Examples C-3
EZACONFG Layout Usage Example C-4
LCA0000 and CFG0000 Control Block Programming Notes C-5

Appendix D: Linking EZASOKET, EZACICSO, and EZACICAL Applications

Linking EZASOKET Applications D-2
Example: Linkage Editor Control Cards D-2
Example: Link Control Cards for IBM TCP Previously Compiled Programs D-3
Linking EZACICSO Applications D-3
Example: EZACICSO Linkage Editor Control Cards D-4
Linking EZACICAL Applications D-4

Appendix E: T09MCALL Macro

Macro Syntax E-1

Index

Chapter

1

Unicenter SOLVE:CPT Tools

The Unicenter® TCPaccess™ Programming Toolkit for CICS, Unicenter® SOLVE:CPT™, is a packaged set of tools to help developers of CICS programs easily use open networking topologies to access data within a wide area network.

The following topics are discussed in this chapter and provide information about the Unicenter SOLVE:CPT Tools:

- Automated Transactions—Describes how to send and receive data over TCP/IP using transient data queues
- The LISTEN Tool—Describes the listening and data processing tasks of the LISTEN tool
- The RECEIVE Tool—Describes how the RECEIVE tool works, including reliability factors
- The SELECT Tool—Describes the SELECT tool which provides pseudo-conversational receive functions
- The SEND Tool—Describes the automated SEND tool and what it does.

Unicenter SOLVE:CPT is comprised of three functional groups:

- CPT Tools
- CPT API services
- CPT Administrative Interface

The Unicenter SOLVE:CPT Administrative Interface

The Unicenter SOLVE:CPT Administrative Interface is a set of CICS transactions that enables you to view everything that is currently occurring within CPT. CPT can facilitate online debugging and analysis of problems with CPT programs. The interface can be used to see current activity, including logging this information statistically for future capacity planning. CPT itself and CPT tools can be dynamically changed and added through these panels. Tracing can be turned on/off; changed; and viewed online to CICS using the administrative interface. Full documentation of the CPT interface is contained in the *Administrator Guide*.

Unicenter SOLVE:CPT API Services

The Unicenter SOLVE:CPT callable API services provide for full duplex (bi-directional) data transfer across a TCP/IP or OSI networks. This set of services implements Berkeley System Development (BSD) compatible sockets. The Unicenter SOLVE:CPT API callable services are a higher-level (smaller) set of verbs enabling TCP and OSI connection functionality. The API provides a higher degree of flexibility when writing an application than using Unicenter SOLVE:CPT tools. The Unicenter SOLVE:CPT API provides full socket capabilities with an easy interface while enabling the same volume and performance abilities normally obtained using the base stack Assembler API. You will find introductory documentation to Unicenter SOLVE:CPT API Services in the chapter “Unicenter SOLVE:CPT API Services.”

The Unicenter SOLVE:CPT FTP Client Callable Service

The Unicenter SOLVE:CPT FTP client callable service provides the ability to send or receive a file to CICS. The FTP service allows a subset of data to be easily read or written to either CICS transient data queue or CICS temporary storage. FTP provides a quick communication between two remote hosts easily accessible to CICS. This is fully compatible with all standard FTP client commands. This service is documented with the Unicenter SOLVE:CPT API services in the chapter “Unicenter SOLVE:CPT API Services.”

Automated Transactions

The automated tools in the Unicenter SOLVE:CPT are pre-written CICS code that CICS programmers can use to send and receive data over TCP/IP using transient data queues.

This feature provides immediate network capabilities with very little development effort. Application programs simply read and write to transient data queues.

Unicenter SOLVE:CPT automated transactions provide a variety of processing options including translation, file or record processing, dynamic queue name resolution, dynamic host and port resolution, buffering requirements, and statistics. These automated transactions can be used in combination with user-written applications or can be used as development tools.

These tools include:

- LISTEN
- RECEIVE
- SEND

The data flow of the automated data transfer transactions is simplex, or in a single direction. This means that the automated data transfer transactions only receive or send data. However, the automated transactions can be used in combination to achieve full-duplex data transfer, or conversational mode, to achieve a wide range of application uses.

Tools Customization

The tools are customized by Unicenter SOLVE:CPT configuration definition statements. There is a specific macro instruction in the CPT configuration for each tool. The configuration macro instructions specify options that control tool transaction processing.

The tools are controlled by CICS table entries and CPT configuration macro statements, which are described in the *Administrator Guide*.

The LISTEN Tool

The LISTEN tool is selected by defining the T09MLSTN macro statement. This configuration macro statement defines an automated listening transaction. The T09MLSTN macro statement contains operands that define well-known port, buffering requirements, statistics, and tracing.

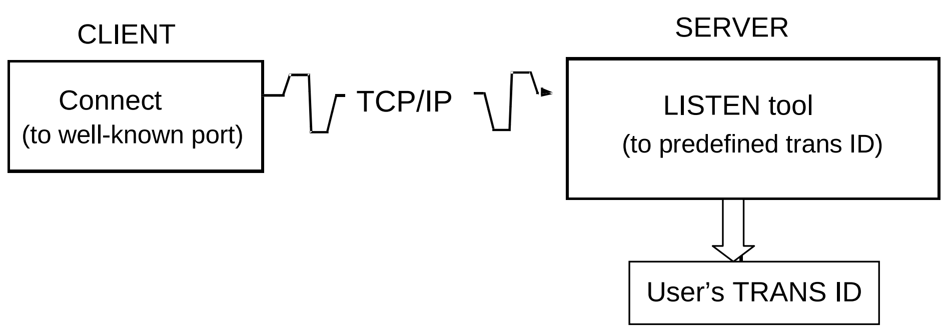
Each TCP server port can be defined, along with its associated data processing transaction ID. The transaction ID can be the RECEIVE tool or a user-written program.

A LISTEN tool can also be defined to let the client determine the transaction ID to start or the transient data queue to write client specified data into.

The LISTEN tool is initiated during CPT startup processing and is defined by an entry in the Unicenter SOLVE:CPT configuration file.

The automated LISTEN tool requires a transaction ID in order for the T09TLSTN program to be specified in the T09MCICS macro instruction. The default listener transaction ID is IPTL. The T09TSTRT program initiates the automated LISTEN tool during CPT initialization. A CICS START command is issued for each T09MLSTN macro instruction configured.

The automated LISTEN transaction provides a mechanism to handle connection establishment and data processing application initialization. The data processing application can be either the Unicenter CPT automated RECEIVE tool or a user-written routine, providing quick development of server applications.



Note: The LISTEN tool is documented in the *Administrator Guide*.

Diagnostics

In order to assist in application development, maintenance, support, and performance tuning, a number of diagnostic tools are provided with LISTEN. When options in the T09MLSTN macro are defined, tracing and statistics can be enabled. Traces are provided for internal flow of control, Unicenter SOLVE:CPT API request arguments, and network data; statistics report the activity on established connections.

MRO Feature

With the MRO feature, the LISTEN tool can be externalized from CICS in order to load balance across several CICS regions, thereby providing CICS storage constraint relief plus all the other benefits provided with running MRO transactions. However, once an endpoint is established in one CICS, it cannot be passed to another CICS.

The RECEIVE Tool

The RECEIVE tool consists of:

- An automated LISTEN transaction
- An automated RECEIVE transaction
- A user-written application

The automated LISTEN transaction can spawn an automated RECEIVE transaction after a client connection is established.

The automated RECEIVE transaction:

- Reads data from the network
- Processes the data
- Writes records to a defined transient data queue

The user-written application is triggered when data is written to the transient data queue according to limits set in the CICS Automatic Transaction Initialization (ATI) facility. These limits are defined in the Destination Control Table (DCT) for the queue.

A single LISTEN transaction can spawn multiple, simultaneous RECEIVE data transactions, thereby servicing multiple clients simultaneously. Hence, the automated RECEIVE tool is a multithreaded server application.

CICS provides initialization of the user application when data is written to the transient data queue. A user application simply reads records from the transient data queue. This provides a transparent network interface for CICS application developers to receive client application information.

Customization

Customization of the automated RECEIVE tool consists of both CICS resource definitions and Unicenter SOLVE:CPT configuration statements. CICS resource definitions are required for the Processing Program Table (PPT), Program Control Table (PCT), and Destination Control Table (DCT). Unicenter SOLVE:CPT macro definition statements are required for the automated LISTEN (T09MLSTN) and RECEIVE (T09MRECV) transactions.

The T09MLSTN macro instructions can specify server information, an automated RECEIVE transaction ID, and a reference to the T09MRECV configuration entry. The default automated RECEIVE transaction ID is IPTR. The T09MRECV macro contains operands specific to the data processing options. The data processing options specify translations, record or file control, transient data queues, statistics, and tracing. The transient data queue name can be identified directly, or dynamically resolved from the first four bytes received from the connection.

Note: The RECEIVE tool is documented in the *Administrator Guide*.

Reliability Factors

The RECEIVE tool provides a mechanism to handle data processing over the open network interface. Stream data received is collected into logical records and written to a transient data queue that can trigger a user-written application. The idiosyncrasies associated with open network data processing are performed by the automated transaction. Additionally, the transient data queue SYNCPOINT and ROLLBACK features are used by the automated RECEIVE transaction to provide reliable data transfer support at the application layer.

The RECEIVE tool is configured with options that control when and how data is written to a transient data queue. These options describe parsing requirements:

- FILE specifies that data is received as a stream and is written to the transient data queue as a single record without parsing
- ALL, LL (Logical Length), and SEP (Separator Errors) specify parsing, and one or more records can be written to the transient data queue

The RECEIVE tool does not explicitly issue SYNCPOINT commands, but rather a SYNCPOINT command is issued by CICS during task termination. However, the CICS ROLLBACK facility is used within the RECEIVE tool when an error in processing is detected.

These general categories classify errors:

- Transport provider errors
- Data processing errors
- Transport Provider Errors

Transport provider errors are determined by a non-zero return code received from a Unicenter SOLVE:CPT API service request. Typical transport provider errors are disconnect indications from a remote host and transport provider termination. Transport provider errors are recorded in the Unicenter SOLVE:CPT error log and are referred to as the diagnostic code. Transport provider errors cause an abortive termination of the connection and cause a CICS SYNCPOINT ROLLBACK command to be issued. The only valid return code that is not considered an error is a release indication, which is interpreted as an end-of-file notification.

Data Processing Errors

A data processing error is determined by a logic error. Typical logic errors are:

- Transient data buffer overflow
- Translation, logical length (LL)
- Separator (SEP) errors

Data processing errors are recorded in the Unicenter SOLVE:CPT error log. Data processing errors cause an abortive termination of the connection and cause a CICS SYNCPOINT ROLLBACK command to be issued.

Diagnostics

A number of diagnostic tools are provided with RECEIVE to assist in application development, maintenance, support, and performance tuning. When options in the T09MRECV macro are defined, tracing and statistics can be enabled. Traces are provided for internal flow of control, Unicenter SOLVE:CPT API request arguments, and network data; statistics report bytes processed, log maximum values, and requests from Unicenter SOLVE:CPT API services.

The SEND Tool

The SEND tool consists of two programs

- The automated SEND transaction that is triggered by the CICS ATI facility.
- A user-written application that is responsible for placing data into a transient data queue. When data is written to a transient data queue, these limits are defined in the DCT for the queue.

The SEND tool is responsible for establishing a connection and processing data. There is no restriction on the number of simultaneously executing client transactions, although you can have only one T09MSEND macro defined for a specific transient data queue, port, and ipname.

A user application is required to write data or records to the transient data queue. The initialization of the automated SEND transaction and the transfer of data is handled transparently. This provides a transparent network interface for CICS application developers to send information to server applications.

Optionally, the SEND tool can be initiated through its transaction ID. The command syntax would contain the transaction ID and a transient data queue name. The SEND tool then process the transient data queue as if initiated by the ATI facility. This provides a restart mechanism for client applications.

Customization

Customization of the automated SEND tool consists of CICS resource definitions and a Unicenter SOLVE:CPT configuration statement. CICS resource definitions are required for the user-written application Processing Program Table (PPT) and the Destination Control Table (DCT). The DCT entry for the transient data queue should specify the T09TSEND transaction ID. The default transaction ID for the T09TSEND program is IPTS. A Unicenter SOLVE:CPT macro definition statement for the automated SEND (T09MSEND) transaction is required.

T09MSEND contains operands specific to the API transport provider and data processing options. These specify fixed or dynamic host name and port resolution, buffering requirements, statistics, and tracing. The data processing options specify translation, record or file control, transient data queue name, statistics, and tracing. The configuration options allow a user to select a fixed server address, or dynamically resolve the server address from the first transient data queue record. T09MSEND contains only one transient data queue name, port, and ipname. As such, multiple T09MSEND macros must be defined in order to address different queues, ports, or ipnames.

Note: The SEND tool is documented in the *Administrator Guide*.

Reliability Factors

The SEND tool provides a mechanism to handle data processing over the open network interface. Data is read from a transient data queue and sent over the connection to a server. The idiosyncrasies associated with open network data processing is handled by the automated SEND transaction. Additionally, the automated SEND transaction uses CICS' transient data queue SYNCPOINT and ROLLBACK features to provide reliable transfer support.

Unicenter SOLVE:CPT automated SEND transactions are configured with options that control how data is sent to the transport provider. These options describe control information required in the data transmission:

- FILE specifies that a connection is established and released for every record read from the transient data queue.
- ALL, LL, and SEP options specify that multiple records can be read from the transient data queue and sent over the connection. The LL and SEP options require control information to be incorporated into the output data.

The SEND tool explicitly issues SYNCPOINT commands after a connection is successfully established and released during FILE option processing. The SYNCPOINT command is not issued while processing ALL, LL, or SEP options, but rather a SYNCPOINT command is issued by CICS during task termination. However, the SEND tool uses the CICS ROLLBACK facility when an error during processing is detected.

These general categories classify errors:

- Transport provider errors
- Data processing errors
- Transport Provider Error

A transport provider error is determined by a non-zero return code received from a Unicenter SOLVE:CPT API service request. Typical transport provider errors are disconnecting indications from the remote host and transport provider termination. Transport provider errors are recorded in the Unicenter SOLVE:CPT error log and are referred to as the diagnostic code. Transport provider errors cause an abortive termination of the connection and a CICS SYNCPOINT ROLLBACK command to be issued. The only valid return code not considered an error is a release indication, which is interpreted as an end-of-file notification.

Data Processing Error

A data processing error is determined by a logic error. Typical logic errors are transient data buffer overflow, translation, logical length (LL) and separator (SEP) errors. Data processing errors are recorded in the Unicenter SOLVE:CPT error log. Data processing errors cause an abortive termination of the connection and a CICS SYNCPOINT ROLLBACK command is issued.

Diagnostics

In order to assist in application development, maintenance, support, and performance tuning, a number of diagnostic tools are provided with SEND. When options in the T09MSEND macro are defined, tracing and statistics can be enabled. Traces are provided for internal flow of control, Unicenter SOLVE:CPT API request arguments, and network data; statistics report bytes processed, log maximum values, and requests from Unicenter SOLVE:CPTAPI services.

See the *Administrator Guide* for information on setting up your network information and enabling these automated tools.

The SELECT Tool

The SELECT tool consists of two programs:

- The automated SELECT transaction
- A user-written application using the RECEIVE API with the ADTNWAIT option and the GIVE API with the AFMOPSEL option.

  The SELECT tool enables the user-written application to be pseudo-conversational. Once the RECEIVE and GIVE calls are performed, the user-written application can perform an EXEC CICS RETURN. Once the data is received, the SELECT tool restarts the transaction to process the data.

In order for the RECEIVE transaction to take advantage of the SELECT tool, it must be coded in a particular way. See the sample program T09PASV5 in T09SAMP for an example of these steps:

1. The ADTNWAIT option flag is specified for ADTOPCD2 in the ADT for RECEIVE. This option tells the RECEIVE service to not issue a wait in the service but instead return to the caller with a CEPWBLCK return code.
2. If the CEPWBLCK return code is returned from RECEIVE, the transaction then calls the GIVE service using the flag AFMOPSEL in the AFMOPDC1 options field. This tells the GIVE service to have the SELECT tool transaction perform the wait.

   Note: If the GIVE service returns CEPESLCT, then the SELECT tool transaction is not running.

   Once the GIVE service completes successfully, the transaction should return to CICS. In addition, the AFM argument can have the next transaction ID placed in AFMNTRAN. This is the transaction that is executed by the SELECT tool once the RECEIVE is complete. It defaults to the current transaction.
3. When the wait finishes for the RECEIVE, the SELECT tool transaction starts the specified transaction. The transaction needs to RETRIEVE the token from the CICS commarea. The length will only be four bytes and is the Unicenter SOLVE:CPT token. Since a receive transaction may also be started by a listener, the program can use the returned length from the RETRIEVE to determine if only the token was passed or an ACM.
4. The newly started transaction now needs to issue the RECEIVE call again. All parameters to the RECEIVE, except for ADTBUFFA, must be the same as when the RECEIVE was issued in Step 1. Various error codes are returned if not or unpredictable results may occur. This RECEIVE may complete with any return code including the CPTWBLCK. If the CPTWBLCK occurs, it returns to Step 2. Otherwise, process the RECEIVE completion.

Note: The SELECT tool is documented in the *Administrator Guide.*

Chapter

2

Unicenter SOLVE:CPT API Services

This chapter provides information about the Unicenter SOLVE:CPT Application Program Interface (API) services.

It discusses the following topics:

- TCP Connection Management—Describes how to use the LISTEN and CONNECT services to provide connection management
- TCP Data Transfer—Describes how to use the SEND and RECEIVE services to provide stream-oriented data transfer
- UDP Data Transfer and Endpoint Creation—Describes how to use the SENDTO and RCVFROM services to establish UDP endpoints and provide datagram transmission capabilities
- Connection and Endpoint Release—Describes how to use the CLOSE service to release a TCP connection, close a UDP endpoint, or optionally shutdown communications to or from a TCP endpoint
- 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
- Unicenter SOLVE:CPT FTP Client Service—Describes how to transfer files from CICS to remote systems using the Internet standard File Transfer Protocol (FTP)
- Security Program—Describes how to invoke and use the optional security program feature provided with the LISTEN service
- Sample Unicenter SOLVE:CPT API Pseudo Code—Provides sample pseudo codes for client and server applications
- Unicenter SOLVE: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
- Using CA-InterTest® with Unicenter SOLVE:CPT Applications—Provides a sample JCL to help you reassemble the CA-InterTest® module to exclude calls to the Unicenter SOLVE:CPT stubs

- Compiling and Linking a CPT API Application—Provides a sample JCL to help you compile and link a Unicenter SOLVE:CPT API Sample Programs

The Unicenter SOLVE:CPT Interface

The intent of Unicenter SOLVE:CPT is to provide the highest level of interface available to the application program without degrading functionality. You can mix and match between using the Tools, FTP client, or API services within CPT. In other words, use the easiest Unicenter SOLVE:CPT service that meets your minimal requirements. A good example of this is using the Unicenter SOLVE:CPT Listen Tool to initiate your application transaction, which then uses Unicenter SOLVE:CPT API services for two way data transfer. In this example, you remove the most complicated code (listen logic) from your program while still maintaining full socket bi-directional capabilities that are desired for a robust TCP/IP program.

A standard set of Berkeley Systems Development™ (BSD) sockets verbs is usually 26 or more verbs. Unicenter SOLVE:CPT reduces this number to eight for your convenience. To allow all the same functionality of BSD sockets with just eight calls to Unicenter SOLVE:CPT requires the passing of a control block that contains the same information as the 26+ BSD verbs. Since most of the fields within these control blocks default quite well, you need only update those parameter settings that affect the way your particular application needs to function.

Implementation of the Unicenter SOLVE:CPT API services is controlled through various subroutine calls. There are internal subroutine calls used to support the Unicenter SOLVE:CPT environment and external subroutine calls used by applications for service requests.

- The internal calls manage resources associated with connections and the *Task-Related User Exit* (TRUE) interface
- The external calls generate service requests related to specific application tasks

The Unicenter SOLVE: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 that show the use of the Unicenter SOLVE:CPT API services at the end of this chapter.

Unicenter SOLVE:CPT Task-Related User Exit Interface (TRUE)

Unicenter SOLVE: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 used by Unicenter SOLVE:CPT is a communication subsystem based on open network protocols. The communication subsystem is an API to a transport provider.

Application Programming Concepts

The Unicenter SOLVE:CPT API facility supports communication with open network protocols using a client/server model. The Unicenter SOLVE:CPT API services are designed to communicate with the transport layer of the Transmission Control Protocol/Internet Protocol (TCP/IP).

A server application passively listens, or waits, for a connection request. Once a connection indication from a client application is 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
- Contacts a well-known port for a server
- 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.

The following sections describe a very high level (condensed version) of pseudo code for writing a typical client and a typical server application using Unicenter SOLVE:CPT API services. Following these sections is a more detailed introduction to each Unicenter SOLVE:CPT API service call. At the end of the chapter are very detailed pseudo code examples for various client and server sample applications.

Client Condensed Pseudo Code

```
...
existing legacy application code to assemble data to be sent
...
initialize CPT API control blocks
call CONNECT CPT API service
call SEND CPT API service
call RECEIVE CPT API service(for acknowledgement record)
call CLOSE CPT API service
CICS RETURN
```

Server Condensed PseudoCode

```
Listen:
 initialize CPT API control blocks
 call LISTEN CPT API service
  (or recommended use the CPT LISTEN Tool)
 call GIVE CPT API service
 CICS EXEC START(mytran)
 Goto Listen:

mytran:
 initialize CPT API control blocks
 call TAKE CPT API service
 call RECEIVE CPT API service
 ...
 existing legacy application code to process the requested service
 ...
 call SEND CPT API service
  to send result of service requester
 call CLOSE CPT API service
 CICS RETURN
```

Note: Multiple mytran transactions will be spawned depending on number of active connections

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) and is used for all subsequent Unicenter SOLVE:CPT service requests related to that connection. Multiple connections or tokens can be gotten 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 gives the TRUE management routines the ability to release resources during normal or abnormal task termination.

Ownership of resources can be controlled:

- Automatically by internal Unicenter SOLVE:CPT routines
- 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, Unicenter SOLVE: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 is 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.

Both the LISTEN and CONNECT services, when used with IBM TCP/IP, have non-blocking open options (set through the ACM) that allow control to be returned immediately to the application program. The SELECT service subsequently can be called to determine if the request was satisfied. This allows for server or client application designs that handle more than one port per CICS transaction.

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 trans ID in the connection management argument or dynamically by the connecting client. You select this option by initializing a field within the connection management argument. Completion of the LISTEN service is generally indicated by an error at Unicenter SOLVE:CPT or transport provider termination.

Note: There is a LISTEN tool available that can be used instead of coding a program to use the LISTEN service. See The LISTEN Tool topic in the chapter "Unicenter SOLVE:CPT Tools."

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 that an ACM 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 representing the established connection with a server. This token is used for all subsequent data transfer, data processing, and connection termination service requests.

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 predetermined mechanism used by both the client and server applications should be designed. Mechanisms such as separator characters, fixed length records, or record header information can be used to delimit records. The Unicenter SOLVE:CPT tools also use these mechanisms.

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 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 or receive data in logical records where the records are separated by a predefined character sequence

Both the SEND and RECEIVE (IBM TCP/IP) services have non-blocking options that are set through the argument for data transfer (ADT) that allow control to be returned immediately to the application program. The SELECT service subsequently can be called to determine if the request was satisfied. This allows for server or client application designs that handle more than one port per CICS transaction.

SEND

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

On 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 that an ADT 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.

Both the SENDTO and RCVFROM (IBM TCP/IP) services have non-blocking options set through the ADT that allow control to be returned immediately to the application program. The SELECT service subsequently can be called to determine if the request was satisfied. This allows for server or client application designs that handle more than one port per CICS transaction.

SENDTO

The SENDTO service is used by a user-written application to send a datagram to a remote UDP endpoint. The SENDTO service requires that an ADT be initialized by the application. It 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 Unicenter SOLVE:CPT SENDTO and RCVFROM 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 that an ADT be initialized by the application. It 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 Unicenter SOLVE:CPT SEND and RECEIVE buffers for the endpoint can be set in the ADT along with optional trace and statistics flags.

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 Unicenter SOLVE:CPT associated resources. Connection Release requires that either a listen or data transfer connection 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 is not 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.

You can use the facility management services 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 that an ACL 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.

The notion of an orderly close in BSD sockets is simply to wait a specified amount of time, so that the other end of the connection can finish receiving data before closing down the connection. This wait or *linger* time can be specified globally through the configuration macro, T09MCICS, or in the ACL when calling the CLOSE service.

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

Optionally, you can use the CLOSE service to implement the BSD Shutdown socket function. This set of options is included in the CLOSE service to accommodate existing applications that depend on this TCP half close mechanism as an application level protocol indicating the closing of a TCP connection.

The available options are:

ACLSHUT0—Disallow RECVS on for this token

ACLSHUT1—Disallow SENDS on for this token

ACLSHUT2—Disable SENDS and RECVS for this token

Calling the CLOSE service with a shutdown option does not close the endpoint or release any associated Unicenter SOLVE:CPT resources.

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 Unicenter SOLVE:CPT system by applying an SMP/E USERMOD. See the *Administrator 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.

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

Facility Management

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

A multithreaded server application is an example of a multitasked application where the Unicenter SOLVE: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 Unicenter SOLVE:CPT connection can benefit from facility management services.

Note: 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.

Unicenter SOLVE:CPT connection management services (LISTEN and CONNECT) create connections. By default, the task that issues a connection management service gets ownership of the connection and its associated resources. Unicenter SOLVE: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 multitasked application can issue a GIVE or TAKE facility management service. The mechanism used to pass information related to a Unicenter SOLVE:CPT connection between tasks is application-dependent.

The IBM IUCV Socket (IBM TCP/IP) interface requires that only one socket function per IUCV path be executed at a time. This adds the requirement of the GIVE and TAKE services to not only manipulate the associations of tokens with CICS transactions, but also with IUCV paths.

The default action for a GIVE service call is to disassociate the token from the caller's CICS task and IUCV path. The default action for the TAKE service is to associate the token with the caller's CICS task and IUCV path. When an application calls the GIVE service with the default action implied, the application subsequently must call the SELECT service to wait for another CICS transaction to successfully TAKE the token that was just given (see the SELECT service).

Note: Applications that want to maintain a TCP connection for a long period, but do not want to use up CICS resources by having the transaction running the entire time, should use the dequeue/enqueue options of the GIVE and TAKE services. These options only disassociate and associate the token from and to a CICS transaction, while maintaining the same IUCV path.

Some applications may benefit by use of the SELECT tool. See the SELECT Tool topic in the chapter "Unicenter SOLVE:CPT Tools" for details.

GIVE

A user-written application uses the GIVE service to disable ownership of internal Unicenter SOLVE:CPT resources associated with a connection. This facility prohibits Unicenter SOLVE: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 multitasked 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 by the CLOSE service after it has been given. The GIVE service only affects implicit release management services provided by the Unicenter SOLVE:CPT task-related user task management routines. In addition, 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.

On 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 get ownership of internal Unicenter SOLVE:CPT resources associated with a connection. This facility enables Unicenter SOLVE: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 that a call 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 was 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 Unicenter SOLVE:CPT TRUE management routines. Hence, the TAKE and, to some extent, GIVE facility management services are optional.

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

Refer to the “Subroutine Calls” chapter for detailed information about the subroutine calls for each language.

Unicenter SOLVE:CPT FTP Client Service

The Unicenter SOLVE:CPT FTP Client Service enables you to transfer files from CICS to remote systems using the Internet standard File Transfer Protocol (FTP).

Unicenter SOLVE:CPT Client FTP offers the following advantages:

- Server FTP support
- Shortened CICS application development time
- Multi-platform availability of a server
- Reliability of a standard application protocol
- Mechanism to simplify FTP client operations

The Client FTP Service provides additional built-in functions that are not available to the other Unicenter SOLVE:CPT tools. While the SEND and RECEIVE tools are easy to use, they still require remote system application development. Unicenter SOLVE:CPT Client FTP applications can take advantage of the FTP server, which is typically a component of an internet-connected host.

Just as for other Unicenter SOLVE:CPT services, client/server architecture, protocols and error recovery are managed by CPT. Application programmers provide minimal information and rely on Unicenter SOLVE:CPT to handle technical issues.

Unicenter SOLVE:CPT Client FTP programs must provide:

- Remote host name or address
- Remote user id and password
- Location of data
- Operation

Depending on the programming language used, information is provided to Unicenter SOLVE:CPT through a common data area, a copybook, or a data structure. A call is issued within the transaction to start the data transfer. Once the transfer is completed, control is returned to the user transaction along with status information about the transfer.

Unicenter SOLVE:CPT Client FTP uses two CICS tasks to accomplish the transfer. One task manages the FTP control connection. The control connection is used to transfer commands that describe the functions to be performed, and to handle the replies to these commands.

These FTP commands:

- Authorize a user
- Specify the parameters for the data connection—transfer mode, representation type, and structure
- Specify file system operations—store, append, rename

The second task manages the data connection that does the actual data transfer. You can use standard defaults for transfers or you can use optional parameters to specify data representation, directory, site commands, mode, and structure. Your application specifies this data within an argument list and calls Unicenter SOLVE:CPT Client FTP to execute the operation using the EXEC CICS LINK command.

Note: Only one file can be transferred with each call, but there is no limit to the number of requests a single transaction can perform.

Unicenter SOLVE:CPT FTP Client Service Overview

The Unicenter SOLVE:CPT FTP Client Service enables Unicenter SOLVE:CPT user applications to use the standard File Transfer Protocol (FTP) to transfer data files from a CICS region to a remote host. The Unicenter SOLVE:CPT FTP Client Service is invoked from CICS programs as a callable service. It requires no end-user interface.

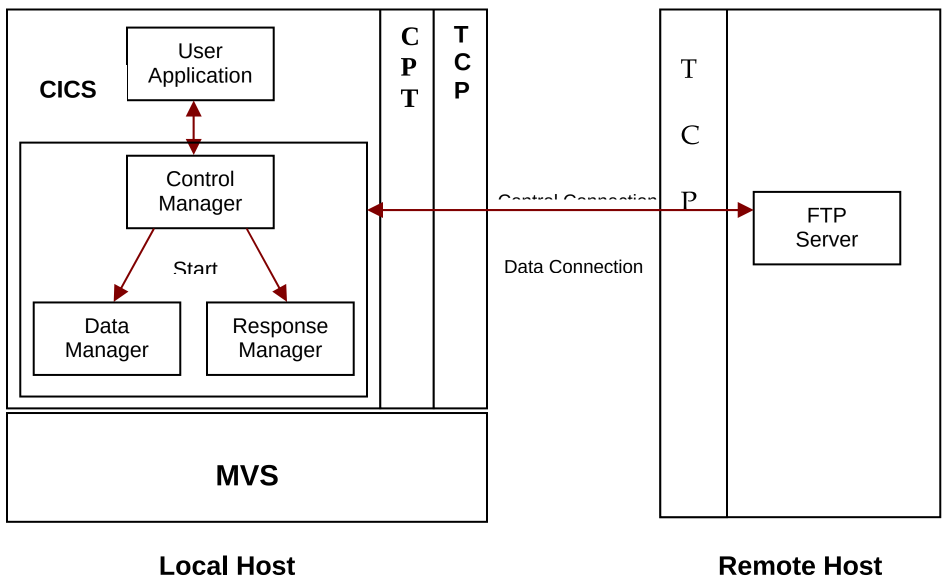
Using parameters passed from the calling application, the Unicenter SOLVE:CPT FTP Client Service:

- Establishes a control connection with the remote host on well-known port 21
- Completes the Telnet remote logon protocol
- Processes any file transfer attributes to the remote server
- Establishes a data connection with the remote server
- Transmits the specified files to the remote server

When the Unicenter SOLVE:CPT FTP Client Service completes this sequence of tasks, the service returns status information to the calling user application in the argument for file transfer (AFT).

Unicenter SOLVE:CPT FTP Client Service Overview shows the functional architecture of the Unicenter SOLVE:CPT FTP Client Service, and the interaction between the Unicenter SOLVE:CPT FTP Client Service and the user applications within the CICS environment.

Unicenter SOLVE:CPT FTP Client Service Architecture



To invoke the Unicenter SOLVE:CPT FTP Client Service, a user application must:

- Create the files to be transferred in a transient-data or temporary-storage queue
- Build an AFT
- Perform an EXEC CICS LINK to the service

The AFT contains information that enables the Unicenter SOLVE:CPT FTP Client Service to locate the files to be transferred and use standard FTP commands to initiate the file transfer.

Security Program

Unicenter SOLVE:CPT provides security through a security program for user evaluation of requests via IP address or User ID/Password for the services of local listeners/servers. If a security program is implemented, the user program is invoked for each connection request. The user program can be specified for each listener if desired. 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 program, the SCTYEXIT=*program-name* must be coded in the T09MCICS macro or the T09MLSTN macro of the T09CONFG Configuration Table. This user program is CICS LINKed during the connection process and must conform to CICS coding standards since you **must** it as a Processing Program Table (PPT) entry.

- If no SCTYEXIT parameter is coded in the Configuration table, all connection requests are authorized and the user ID will be the same as the Listener transaction
- If SCTYEXIT is coded but the program is missing or is disabled, no connections is permitted

Note: Invoking the Administrator Interface panel for the Configuration Table can check the second condition. However, the security program is displayed only if it is disabled or if it is not in the PPT.

Each Listen tool or user-written listener can specify its own security program. If the Listen tool or the user written listener does not specify a security program and the SCTYEXIT parameter is coded on the T09MCICS configuration macro, then that program is used as the security program.

Typically, a security program is called only when either the ACMTRNID is specified or a user-written listener sets the ACMLTRAN option. However, by coding SCTYTYPE=MANDTORY and SCTYEXIT=*program-name* on the T09MCICS configuration macro, the security program will be executed before returning control to the user-written listener.

Normally a security program is invoked only when a server transaction is automatically started within the Unicenter SOLVE:CPT Listen service as a result of one of the following:

- The transaction was specified in ACMTRNID
- The transaction was dynamically obtained from Client Data (ACMOPTNS = ACMLTRAN and ACMTIMEO >0)
- A T09MLSTN macro for the Listen Tool specified either a TRANID parameter or a CLNTIME parameter

In other client/server designs, the application receives control when the connection is made and should make any desired security checks before beginning server activity. However, by coding SCTYTYPE=MANDTORY and SCTYEXIT=*program-name* on the T09MCICS configuration macro, the security program is executed before returning control to the application.

Security Program

The user security program is responsible not only for making the determination of whether a connection is authorized, but also for any desired logging or other capture of unauthorized requests. Because the program is driven for each connection on a listener, performance implications should be considered in designing security programs.

When security is specified in the Configuration table, a new transaction is started (the program is T09TLST2 with transaction ID IPT2). This transaction then CICS links to the specified security program. The program is passed the Security Communications Block (SCB). It contains fields used to determine the validity of the connection. One of the fields in the SCB is the token of the connection. The token can be used to initiate SEND and RECEIVE calls in order to communicate with the remote client to determine a user ID, password, or any other identifying characteristics. Any of the other fields in the SCB may be used as well.

Upon return from the security program, four fields are used from the SCB:

- The authorization switch authorizes the connection by setting a character 1 in the field
- The terminal facility specifies a CICS term ID to associate with the new transaction to be STARTed
- If the user ID field is specified, the new transaction is STARTed with that user ID
- The transaction to be started can also be modified by the security program and then that specified transaction is STARTed

Note: When term ID and user ID are specified, any CICS security for the term ID and user ID are in effect.

The security program can perform additional SEND and RECEIVE calls to request and retrieve data. This data might be some form of user ID or password. The program could then verify the user ID and password with the EXEC CICS VERIFY command. If the user ID is returned in the SCB, the new transaction is started with EXEC CICS START USERID (user ID).

The Security Communications Block

The connection process transaction and the user security program communicate through the Security Communications Block (SCB). Unicenter SOLVE:CPT provides information about the request and its origin. The user security program determines whether the request is authorized and, optionally the name of a terminal facility or user ID to associate with a STARTed server transaction. A DSECT of the SCB for Assembler programs may be generated with the T09DSCTY macro.

This is what the T09DSCTY DSECT control block looks like in Assembler language:

```
Name Operation Operands Description

SECPARM DSECT
SECTRAN DS CL4  SERVER TRANSACTION REQUESTED
SECDATA DS XL40  REQUESTOR DATA
SECSTRT DS CL2  HOW TASK IS TO BE STARTED
SECICTM DS XL6  INTERVAL CONTROL TIME
SECADRS DS 0CL8  REQUESTOR ADDRESS
SECAFAM DS H  DOMAIN
SECRPRT DS H  PORT
SECRHST DS F  HOST IP ADDRESS
SECACTN DS CL1  PERMIT/PROHIBIT SWITCH
SECPRMT EQU C'1'  ..OKAY, INITIATE TASK
  DS X  RESERVED
SECTMID DS CL4  ANY ASSOCIATED CICS TERMINAL
SECLPRT DS H  LOCAL SERVER PORT
SECUSER DS CL8  USER ID
  DS CL512  RESERVED
SECTOKN DS F  TOKEN - ENDPOINT
SECLHST DS F  LOCAL HOST
*
SECLEN EQU *-&LABEL LENGTH OF SECURITY DATA AREA
```

Security Communications Block

Field	Format	Description
SECTRAN	4-byte character	Requested server transaction, maybe modified by the program.
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	2-byte binary	Address family: Inet domain=2.
SECRPRT	2-byte binary	Client remote port number.
SECRHST	4-byte binary	Client remote host IP address.
SECACTN	1-byte character	Authorization switch: ■ 1=accept ■ 0=fail
SECTMID	4-byte character	Associated terminal facility.
SECLPRT	2-byte binary	Requested server local port.
SECUSER	8-byte binary	Returned user ID
SECTOKN	4-byte binary	Token that represents the TCP connection.
SECLHST	4-byte binary	Local host IP address.

Sample Unicenter SOLVE:CPT API Pseudo Code

This section provides 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.
- 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 is 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.
```

```
.
. 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.
.
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
```

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 is terminated. Therefore, this is a single-threaded server application. The application loops within the Unicenter SOLVE:CPT receive/send logic until a Unicenter SOLVE:CPT release indication is determined and then closes the connection gracefully.

Note: Any unexpected error while receiving and sending data causes the connection to terminate abnormally.

The LISTEN service request returns two tokens:

- One token represents the data transfer connection—used with send and receive processing
- The other token represents the server connection—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.
.
. 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
```

Server Application Example 2

This example shows a multithreaded CICS server application where 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 is 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.

Note: 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.
.
 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 shows a data processing program associated with a multithreaded server application. A server program initiates the transaction after a client connection is 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 Unicenter SOLVE:CPT receive/send logic until a Unicenter SOLVE: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.
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
```

Server Application Example 4

This example is a variation of the multithreaded 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 occurs. The error could be either Unicenter SOLVE:CPT and CICS termination, or some unexpected error. Unicenter SOLVE: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
```

Unicenter SOLVE:CPT API Sample Programs

These sample programs are in the T09SAMP data set that was unloaded when Unicenter SOLVE:CPT was installed. Descriptions of each program are provided below.

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

T09SAMP Member Name	Language	Type
T09PACL1	Assembler	TCP Client 1 program. Client Application sends typed in data to the server waiting for the information to be echoed back from the server.
T09PACL2	Assembler	TCP Client 2 program. Client Application to send an internal message using either the FULL, SEP or LL to be echoed back by the server.
T09PAFTP	Assembler	FTP Client Application.
T09PCFT3	Assembler	FTP Client that uses transient data queues.
T09PASV1	Assembler	TCP Server 1 program is a single-threaded server using a Listen API call.
T09PASV2	Assembler	TCP Server 2 program is a multithreaded server using the Listen tool.
T09PASV3	Assembler	TCP Server 3 program is multi-threaded server using a Listen API call with an independent EXEC CICS START tran.
T09PASV4	Assembler	TCP Server 4 program is a multi-threaded server using a Listen API call that has CPT internally issuing the EXEC CICS START tran.
T09PASV5	Assembler	TCP Server 5 program is a multithreaded server using both the Listen and SELECT tools.
T09PACLU	Assembler	UDP Client program.

T09SAMP Member Name	Language	Type
T09PASVU	Assembler	UDP Server program.
T09PCCL1	COBOL	TCP Client 1 program. Client Application sends typed in data to the server waiting for the information to be echoed back from the server.
T09PCCL2	COBOL	TCP Client 2 program. Client Application to send an internal message using either the FULL, SEP or LL to be echoed back by the server.
T09PCFTP	COBOL	FTP Client Application.
T09PCSV1	COBOL	TCP Server 1 program is a single-threaded server using a Listen API call.
T09PCSV2	COBOL	TCP Server 2 program is a multithreaded server using the Listen tool.
T09PCSV3	COBOL	TCP Server 3 program is multi-threaded server using a Listen API call with an independent EXEC CICS START tran.
T09PCSV4	COBOL	TCP Server 4 program is a multi-threaded server using a Listen API call that has CPT internally issuing the EXEC CICS START tran.
T09PCSV5	COBOL	TCP Server 5 program is a multithreaded server using both the Listen and SELECT tools.
T09PCCLU	COBOL	UDP Client program.
T09PCSVU	COBOL	UDP Server program.
T09PPCL1	PL/I	TCP Client 1 program. Client Application sends typed in data to the server waiting for the information to be echoed back from the server.

T09SAMP Member Name	Language	Type
T09PPCL2	PL/1	TCP Client 2 program. Client Application to send an internal message using either the FULL, SEP or LL to be echoed back by the server.
T09PPFTP	PL/I	FTP Client Application
T09PPSV1	PL/I	TCP Server 1 program is a single-threaded server using a Listen API call.
T09PPSV2	PL/I	TCP Server 2 program is a multithreaded server using the Listen tool.
T09PPSV3	PL/I	TCP Server 3 program is multi-threaded server using a Listen API call with an independent EXEC CICS START tran.
T09PPSV4	PL/I	TCP Server 4 program is a multi-threaded server using a Listen API call that has CPT internally issuing the EXEC CICS START tran.
T09PPCLU	PL/1	UDP Client program.
T09PPSVU	PL/1	UDP Server program.
T09PSCL1	C	TCP Client 1 program. Client Application sends typed in data to the server waiting for the information to be echoed back from the server.
T09PSCL2	C	TCP Client 2 program. Client Application to send an internal message using either the FULL, SEP or LL to be echoed back by the server.
T09PSFTP	C	FTP Client Application
T09PSSV1	C	TCP Server 1 program is a single-threaded server using a Listen API call.

T09SAMP Member Name	Language	Type
T09PSSV2	C	TCP Server 2 program is a multithreaded server using the Listen tool.
T09PSSV3	C	TCP Server 3 program is multi-threaded server using a Listen API call with an independent EXEC CICS START tran.
T09PSSV4	C	TCP Server 4 program is a multi-threaded server using a Listen API call that has CPT internally issuing the EXEC CICS START tran.
T09PSCLU	C	UDP Client program.
T09PSSVU	C	UDP Server program.

Note: The *x* in the fifth character in the program name denotes the programming language from the table shown above.

Client 1 Sample Program

T09P*x*CL1 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 is 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
- 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

T09P*x*CL2 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 Unicenter SOLVE:CPT to format the stream data into logical records. These SEND and RECEIVE options make logical record programming much easier from the Unicenter SOLVE: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. T09P*x*CL2 sends the data to the TCP Echo server.

TCP Server 1 Sample Program

T09P*x*SV1 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, Unicenter SOLVE: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

T09P*x*SV2 is an example of a server program that does not issue a listen, but takes the connection from the original listener. Unicenter SOLVE:CPT initiates it when a listening task detects a client request for the port number assigned to this server.

CPTP*x*SV2 can be initiated directly by:

- Another transaction that is a listening server
- By Unicenter SOLVE:CPT from a listening transaction's specification of ACMTRNID in its connection management argument
- A listener specified in a T09MLSTN statement in the Unicenter SOLVE: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

T09P*x*SV3 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, T09P*x*SV3 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 Unicenter SOLVE:CPT is stopped.

Server 4 Sample Program

T09P*x*SV4 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 Unicenter SOLVE: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 Unicenter SOLVE:CPT is stopped.

Server 5 Sample Program

T09PASV5 is an example receive program that uses the SELECT tool.

UDP Client Sample Program

T09P*x*CLU 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 T09P*x*SVU being the other possible destination by specifying the associated transaction ID. When the datagram is received back from the RCVFROM service, the sample client closes the endpoint.

UDP Server Sample Program

T09P*x*SVU 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.

Using CA-InterTest® with Unicenter SOLVE:CPT Applications

In order to use CA-InterTest on application modules using Unicenter SOLVE:CPT stub calls, the InterTest module, IN25UEXI, must be reassembled with entries to exclude calls to the Unicenter SOLVE:CPT stubs.

The following sample JCL can be modified to meet your system requirements. See the CA-InterTest *MVS Installation and Customization Guide* for a discussion on the IN25UEXI exit.

```
//IN25UEXI JOB ...

//ASM  EXEC PGM-IEV90,REGION=102K,
//  PARM='DECK,LIST,XREF(SHORT),ALIGN'
//SYSPRINT DD SYSOUT=A
//SYSPUNCH DD DSN=&&LOADSET,DISP=(NEW,PASS),UNIT=SYSDA,
//  DCB=(RECFM=FB,LRECL=80,BLKSIZE=400,SPACE=(400,(100,100,1))
//SYSLIB DD DSN=CAI,SAMPLIB,DISP=SHR
//SYSUT1 DD UNIT=SYSDA,SPACE=(CYL,(5,1))
//SYSUT2 DD UNIT=SYSDA,SPACE=(CYL,(5,1))
//SYSUT3 DD UNIT=SYSDA,SPACE=(CYL,(5,1))
//SYSIN DD *
 IN25UEX CALL=T09FCLOS
 IN25UEX CALL=T09FXLAT
 IN25UEX CALL=T09FCONN
 IN25UEX CALL=T09FGIVE
 IN25UEX CALL=T09FLSTN
 IN25UEX CALL=T09FRCFR
 IN25UEX CALL=T09FRECV
 IN25UEX CALL=T09FSEND
 IN25UEX CALL=T09FSLCT
 IN25UEX CALL=T09FNTO
 IN25UEX CALL=T09FTAKE
*
* INSERT YOUR IN25UEX STATEMENTS FOR SPECIAL CALLS HERE
*
 IN25UEX TYPE=FINAL
 T09FCLOS DC XL16'90ECD00C183F4510300E0180000058F0'
 T09FCONN DC XL16'90ECD00C183F4510300E0180000058F0'
 T09FGIVE DC XL16'90ECD00C183F4510300E0180000058F0'
 T09FLSTN DC XL16'90ECD00C183F4510300E0180000058F0'
 T09FRCFR DC XL16'90ECD00C183F4510300E0180000058F0'
 T09FRECV DC XL16'90ECD00C183F4510300E0180000058F0'
 T09FSEND DC XL16'90ECD00C183F4510300E0180000058F0'
 T09FSLCT DC XL16'90ECD00C183F4510300E0180000058F0'
 T09FSNTO DC XL16'90ECD00C183F4510300E0180000058F0'
 T09FTAKE DC XL16'90ECD00C183F4510300E0180000058F0'
 T09FXLAT DC XL16'90ECD00C183F4510300E0180000058F0'
```

```
*
* INSERT ANY USER WRITTEN ROUTINE HERE
*
 END TERMINATEST THE ASSEMBLY OF IN25UEXI
/*
//SYSUT3 DD UNIT=SYSDA,SPACE=(CYL,(5,1))
//LKED EXEC PGM=IEWL,REGION=512K,PARM=(XREF,LIST,MAP)
*
* INSERT ANY //SYSLIB STATEMENT FOR SPECIAL LOADERS HERE
*
//SYSLMOD DD DSN=CAI,CACIGSxx,DISP=SHR
//SYSUT1 DD UNIT=SYSDA,DCB=BLKSIZE=1024,SPACE=(1024,(200,200))
//SYSPRINT DD SYSOUT=A
//SYSLIN DD DSN=&&LOADSET,DISP=(OLD,DELETE)
//  DD *
 ENTRY IN25UEXI
 NAME IN25UEXI(R)
//
```

Note: For the SYSLMOD DD statement, replace *xx* with the CICS release number. For example, 41 for CICS 4.1.

Compiling and Linking a CPT API Application

There are two simple updates to your existing CICS application compilation and linking JCL that are needed to support CPT/API applications:

1. In your compile step add the following DD:
   `cpthlq.T09MAC`

   The T09MAC library contains all the Assembler macros, Assembler DSECTs, COBOL copybooks, C and PL/I structures needed when compiling your CPT/API application

2. In your link step add the following DD:

   `cpthlq.T09LOAD`

   The T09LOAD library contains all the T09F*xxxx* CICS TRUE stubs that are called by your CPT/API application in order to use CPT service calls.

   Note: The T09LOAD library also needs to be in your CICS startup JCL as part of the DFHRPL concatenation.

Chapter 3

CLOSE Service

Closes an established connection. Both orderly (graceful) and abortive termination options are supported. CLOSE performs all associated functions required for Unicenter SOLVE:CPT resource clean up.

To invoke the CLOSE service, a user application must first build an Argument for Close (ACL) and then 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 chapter discusses the following topics:

- Call Syntax—Shows sample syntax for the CLOSE service call
- Recommended ACL Parameters—Lists the parameters normally used and recommended for the CLOSE service call
- Usage Example—A sample shell of a program using the CLOSE service call
- Parameter Values Returned in the ACL—List the fields that are updated in the ACL control block upon return from the CLOSE service call
- Assembler DSECTS—Provides a list and information about the distributed sample Assembler DSECTS that are used by the CLOSE service call
- Sample Programs—Lists and describes the distributed sample Assembler programs that are use the CLOSE service call along with other service calls.
- Completion Information—Describes the expected results at completion of the CLOSE service call
- Return Codes—Provides a list of return codes that can apply to the CLOSE service call
- Usage Notes—Contains miscellaneous notes about usage of the CLOSE service call
- Complete Parameter List—Provides a complete list of all the parameters and options of those parameters for the CLOSE service call

Call Syntax

We recommend that a site use the T09MCALL macro to call CLOSE:

```
LA    R01,CPTACL
ST    R01,CPTPARMS
T09MCALL CLOSE,PARM=CPTPARMS
```

However, a programmer can call the CLOSE interface stub directly:

```
LA    R01,CPTACL
ST    R01,CPTPARMS
LA    R01,CPTPARMS
L     R15,=V(T09FCLOS)
BALR  R14,R15
```

Recommended ACL Parameters

The following list contains the recommended parameters to use with the CLOSE service. These parameters are set within the ACL control block. See Assembler DSECTS for sample information.

For a complete list of optional parameters, see Complete Parameter List.

Field name	Description
ACLOPCD1	Set to ACLORDER indicating a graceful termination and implements orderly release of the TCP/IP connection.
ACLTOKEN	Connection or endpoint token.
ACLVERS	Version number set to two (2).

Usage Examples

In the following examples, a subset of the actual statements required is shown to emphasize the use of a CLOSE call. It is always recommended that you use the graceful approach unless a network error makes it a requirement to abort the connection.

Graceful Close	Both ends of a session follow the FIN, ACK-FIN shutdown process to terminate a session.
Abortive Close	A reset is generated terminating the session. When errors show that a connection may be corrupt, an abortive close always works.

Important! *Data may be lost when an abortive close is used.*

For a reference to a more complete sample, see Sample Programs.

Graceful Close

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 Unicenter SOLVE:CPT service requests. The token is set before issuing the CLOSE call. 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.

Note: The statements related to the example are in **bold**.

```
*  Dsect's

  T09DACM MF=DSECT                    Argument for Connection Management
  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 initialization and request
           .
           L    R9,ACMTOKEN            Load ACM Token
           .
           . Application and CPT Data Transfer (SEND/RECEIVE) processing
           .
*
```

```
*  CPT Connection Termination
CLOSE     DS     0H
          USING  ACL,CLOSEARG
*
          ST     R9,ACLTOKEN        Save connection token
*
          LA    R01,CLOSEARG        Point to control block
          ST    R01,CPTPARMS        Store address of control block
          T09MCALL CLOSE,PARM=CPTPARMS
          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
```

Abortive Close

This example establishes a connection, receives an error while processing data and aborts the connection. The ACL version and token are specified. The ACL abort option ACLABORT 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.

Note: The statements related to the example are in **bold**.

```
*   Dsect's

   T09DACL MF=DSECT                  Argument for Connection Release
*
*  Working storage
DFHEISTG  DSECT
CLOSEARG  DS     XL(ACLLEN)           Argument for Connection Release
*
*         Entry
label     DFHEIENT
           .
           .           .
          L      R9,ACMTOKEN          Load ACM Token
           .
           . Application processing
           .
DTERROR   DS     0H
*
          USING  ACL, CLOSEARG
*
          ST     R9,ACLTOKEN           Save connection token
          MVI    ACLOPCD1,ACLABORT     Set Abortive
          *
          LA    R01,CLOSEARG         Point to control block
          ST    R01,CPTPARMS        Store address of control block
          T09MCALL CLOSE,PARM=CPTPARMS
*
          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
```

Parameter Values Returned in the ACL

After the CLOSE service call returns control to your application program, the following fields are propagated with connection termination information. These updated values are passed back to the application in the ACL control block.

Field Name	Description
ACLRTNCD	Return code.
ACLDGNCD	Diagnostic code.

Assembler DSECTS

Sample Assembler DSECTS are provided in the distributed software and are available to you in ***cpthlq***.T09MAC. Variable field names contained in the distributed samples and the examples in this guide refer to these DSECTS.

T09DACL	Assembler DSECT name for the ACL. For detailed information and a sample copy of the Assembler DSECT, see the ACL: Argument for CLOSE Used by the CLOSE Service section in appendix “Control Block Layouts”.

All Assembler constants that apply to ACL calls are imbedded in the ACL DSECT sample.

Sample Programs

Sample Assembler source code is provided for your use. You should be able to find a sample that matches your programming requirement. For more complete details on which functions a sample program provides, read the program descriptions in the "Unicenter SOLVE:CPT API Services" chapter and the comments at the beginning of the sample members listed below. These sample program members are available in the distributed software in the ***cpthlq***.T09SAMP library.

Name	Description
T09PACL1	TCP Client 1 program.
T09PACL2	TCP Client 2 program.
T09PASV1	TCP Server 1 program is a single-threaded server using a Listen API call.
T09PASV2	TCP Server 2 program is a multithreaded server using the Listen tool.
T09PASV3	TCP Server 3 program is multithreaded server using a Listen API call with an independent EXEC CICS START tran.
T09PASV4	TCP Server 4 program is a multithreaded server using a Listen API call that has CPT internally issuing the EXEC CICS START tran.
T09PASV5	TCP Server 5 program— Is like SV2 and uses the Select tool.

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 SEND and RECEIVE requests to complete and then waits for both ends of the full-duplex connection to close. This waiting can last up to the number of seconds specified by the ACLTIMEO linger value.

Abortive termination closes the transport provider connection without regard to pending transport provider requests.

WARNING! An abortive termination may 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 register zero (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 zero, indicate the nature of the failure. The diagnostic code may contain a specific code identifying a particular transport provider error. The diagnostic code is normally referred as the error number or just ERRNO, and can be referenced as any EZASOKET call ERRNO.

Return Codes

The CLOSE service returns a code in registers R15 and R0 indicating the results of the execution. These values are in the ACMRTNCD (R15) and ACMDGNCD (R0) described in the appendix “Return Codes.” The diagnostic code may be an ERRNO, CICS abend code or other value depending on the return code.

This DSECT is available in the distributed software in ***cpthlq***.T09MAC in member T09DRTCD. See the appendix “Return Codes” for a sample copy of the T09DRTCD DSECT. Contained in this DSECT is a description of the problem causing the associated return code.

The following is a list of return codes that can apply to the CLOSE call.

Decimal	Hex	Diagnostic Code	Variable	Description
0	0	No	CPTIRCOK	Request completed successfully.
17	11	No	CPTEVERS	Control block version number not supported.
20	14	No	CPTETOKN	Specified data transfer token is invalid.
24	18	No	CPTECOPT	Invalid Close mode specification.
31	1F	No	CPTEFRMT	Other Socket Call Parameter List format or specification error.
34	22	No	CPTENAPI	API not fully available; retry.
40	28	Yes	CPTETERM	Environment is being terminated.
47	2F	Yes	CPTEENVR	Other TPL environmental condition.
65	41	Yes	CPTERLSE	Orderly release of remote connection request.
68	44	Yes	CPTEDISC	Remote connection not available or aborted.
		Yes	CPTEINTG	Transport provider API integrity error.
72	48	Yes	CPTEPRGE	Remote connection environment terminating.
143	8F	Yes	CPTEPROC	Procedural error.
254	FE	Is abend	CPTABEND	Abnormal termination.

Decimal	Hex	Diagnostic Code	Variable	Description
		code		**Note:** The diagnostic code is the abnormal termination code, which is normally a CICS abend code, but can also be in the "Abend Codes" chapter of the *Message Guide*.
255	FF	No	CPTEOTHR	Other error.

Usage Notes

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 uses 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 one before the CLOSE service.

The version number, ACLVERS, indicates the Unicenter SOLVE:CPT release level in which this user application program is written. This required field must be set to two (2) and is validated by the CLOSE service before it processes the request.

The function code, ACLFUNC, indicates the Unicenter SOLVE: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 will 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, only two options (ACLORDER and ACLABORT) are supported; no facility exists for CLOSE event notification, except by way of return code values.

The notion of an orderly close in BSD sockets is to wait a specified amount of time, so that the other end of the connection can finish receiving data before closing down the connection. This is what the ACLTIMEO value is used for by the CLOSE service when the ACLORDER option is specified.

If the option code ACLORDER is selected, the CLOSE service performs a graceful termination. A graceful termination waits for all pending transport provider SEND and RECEIVE requests to complete and then waits for both ends of the full-duplex connection to close. This waiting lasts for the linger value before closing the connection. The linger value is defined by the value in the ACLTIMEO field. This may require the CLOSE service to block the application. This option then performs an orderly release of the TCP/IP connection.

Graceful termination using the ACLORDER option 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 receives a disconnect indication.

WARNING! *An abortive termination may cause data loss and should be used only when data integrity is not required.*

Complete Parameter List

Note: For a recommended list of parameters, see Recommended ACL Parameters earlier in this chapter.

ACLDGNCD	Diagnostic code. Indicates the diagnostic code set by the service request. This value generally indicates a transport provider return code. 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.
ACLOPCDS	Specifies CLOSE processing control options. Supported options:

ACLABORT	Indicates abortive termination and option implements a disconnect or reset of the TCP/IP connection. Typically, used after an unrecoverable application error occurs. *WARNING! An abortive termination may cause data loss and should be used only when data integrity is not required.*
ACLFCLSE	Internal use only.
ACLFFREE	Internal use only.
ACLORDER	Indicates a graceful termination and implements orderly release of the TCP/IP connection. **Note:** This is the preferred option for terminating a connection.
ACLFPURG	Internal use only.
ACLSHUT0	Not currently supported. Shutdown the socket for RECEIVES. If ACMSTATS was set to ACMSTERM, a message is generated.
ACLSHUT1	Not currently supported. Shuts down the socket for SENDS. If ACMSTATS was set to ACMSTERM, then a message is generated.

ACLSHUT2	Not currently supported. Shutdown the socket for RECEIVES and SENDS. If ACMSTATS was set to ACMSTERM, then a message is generated.

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. Unicenter SOLVE:CPT knows if the token is UDP and closes it properly.

Default: ACLORDER.

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: None.

ACLTIMEO

Specifies the time to wait (linger) on an orderly (ACLORDER) CLOSE request. This orderly close is also known as a graceful termination. If this value is not specified on an orderly CLOSE request, the value specified with the LINGER= keyword on the T09MCICS configuration macro within the Solve: configuration table is used. For more information about setting the linger value, see the "Configuration Reference" chapter of the *Administrator Guide*.

Default: One (1).

ACLTOKEN

Required. 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.

Default: None.

ACLVERS

Required. Version number. Indicates the Unicenter SOLVE:CPT version number of the argument list used by the calling program.

Note: Must be set to a binary two for this release of Unicenter SOLVE:CPT.

Default: None.

Chapter 4

CONNECT Service

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 argument for connect management (ACM).

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

This chapter discusses these topics:

- Call Syntax—Shows sample syntax for the CONNECT service call
- Recommended ACM Parameters—Lists the parameters typically used and recommended for the CONNECT service call
- Usage Example—Provides a sample shell of a program using the CONNECT service call
- Parameter Values Returned in the ACM—Lists the fields that are updated in the ACM control block upon return from the CONNECT service call
- Assembler DSECTs—Provides a list and information about the distributed sample Assembler DSECTs used by the CONNECT service call.
- Sample Programs—Lists and describes the distributed sample Assembler programs that use the CONNECT service call along with other service calls
- Completion Information—Describes the expected results at completion of the CONNECT service call
- Return Codes—Lists the return codes that can apply to the CONNECT service call
- Usage Notes—Provides miscellaneous notes about the CONNECT service call usage
- Complete Parameter List—Provides a complete list of the parameters and their options for the CONNECT service call parameters and their options

Call Syntax

We recommend that a site use the T09MCALL macro to call CONNECT:

```
LA    R01,CPTACM
ST    R01,CPTPARMS
T09MCALL CONNECT,PARM=CPTPARMS
```

However, a programmer can call the CONNECT interface stub directly:

```
LA    R01,CPTACM
ST    R01,CPTPARMS
LA    R01,CPTPARMS
L     R15,=V(T09FCONN)
BALR  R14,R15
```

Recommended ACM Parameters

The following table lists the recommended parameters to use with the CONNECT service. These parameters are set within the ACM control block; see the Assembler DSECTs for sample information.

For a complete list of optional parameters, see Complete Parameter List.

Field Name	Description
ACMOPTNS	Connection Initialization Options—Set to ACMNODNR.
ACMRADDR	Remote IP Host Address in hexadecimal.
ACMRPORT	Remote Well-Known Service Port.
ACMVERS	Version number should be set to 2. The T09MCALL macro automatically sets the correct version number in the ACM parameter list.

A character text dot format IP address such as 123.234.123.234 can be set in the ACMRNAME field rather than using the ACMRADDR field.

Usage Example

In the example listed below, a subset of the actual statements required is shown to emphasize the use of a CONNECT call. In this example, a simple client ACM is built and the CONNECT request is performed. The application program sets the remote server port number to 1234. The remote IP address to connect to is placed in the ACMRNAME field. The T09MCALL Assembler macro instruction sets the argument version number and calls 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 ACMTOKEN is used for all Unicenter SOLVE:CPT requests related to the connection.

Note: The statements related to the example are in **bold**.

```
*   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
*
         USING  ACM,CONNARG
*
         MVI   ACMOPTN1,ACMNODNR           No DNR calls
         MVC   ACMRPORT,=H'1234'           Set Server well-known port
         MVC   ACMRNAME(15),=C'123.234.123.234'  Set Server Host name
         LA    R01,CONNARG                 Point to control block
         ST    R01,CPTPARMS                Store address of control block
         T09MCALL CONNECT,PARM=CPTPARMS     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
```

Parameter Values Returned in the ACM

After the CONNECT service call returns control to your application program, the following fields are propagated with valid established connection information. These updated values are passed back to the application in the ACM control block.

Field Name	Description
ACMDGNCD	Diagnostic Code.
ACMLADDR	Local IP Host Address.
ACMLNAME	Local IP Host Name.
ACMLPORT	Client Application Port.
ACMMRECV	API receive buffer size.
ACMMSEND	API send buffer size.
ACMQRECV	API receive queue size, set to one.
ACMQSEND	API send queue size, set to one.
ACMRADDR	Remote IP Host Address.
ACMRTNCD	Return Code.
ACMTOKEN	Token—Connection or endpoint.

Assembler DSECTs

Sample Assembler DSECTs are provided in the distributed software and are available to you in ***cpthlq***.T09MAC. Variable field names contained in the distributed samples and the examples in this guide refer to these DSECTs.

T09DACM	Assembler DSECT name for the ACM. For detailed information and a sample copy of the Assembler DSECT, see the ACM: Argument for Connection Management Used by CONNECT and LISTEN Calls section in the appendix "Control Block Layouts."

All Assembler constants that apply to ACM calls are imbedded in the ACM DSECT sample.

Sample Programs

Sample Assembler source code is provided for your use. You should be able to find a sample that matches your programming requirement. For more complete details on which functions a sample program provides, see the program descriptions in the "Unicenter SOLVE:CPT API Services" chapter and the comments at the beginning of the sample members listed below. These sample program members are available in the distributed software in the ***cpthlq***.T09SAMP library.

Name	Description
T09PACL1	Client application sends typed in data to the server waiting for the information to be echoed back from the server.
T09PACL2	Client application to send an internal message using either the FULL, SEP or LL to be echoed back by the server.

Completion Information

The CONNECT service completes normally when a connection with a remote server is established. The CONNECT service initializes the client environment with the transport provider (API) and actively contacts a remote server and then updates connection information within the ACM control block.

When a connection is successfully established the ACM control block is updated with information related to the connection. The local and remote port, IP address, and host names are resolved and available in the ACM. 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. When Unicenter SOLVE:CPT successfully establishes a client connection a non-zero token will be returned in the ACMTOKEN field. This token can be passed in subsequent Unicenter SOLVE:CPT calls (SEND, RECEIVE, GIVE, and so on) in the token field.

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 returns a code in registers R15 and R0 indicating the results of the execution. These values are in the ACMRTNCD (R15) and ACMDGNCD (R0) described in the appendix "Return Codes." The diagnostic code typically indicates the transport provider return code, better known as Error Number or ERRNO.

A sample Assembler DSECT is provided in ***cpthlq***.T09MAC, in member T09DRTCD. It details the variable field names contained in the distributed samples and the examples in this guide. See the appendix "Return Codes" for a sample copy of the T09DRTCD DSECT. A description of the problem causing the associated return code is contained in this DSECT.

The following table lists return codes that can apply to the CONNECT call.

Decimal	Hex	Diagnostic Code	Variable	Description
0	0	No	CPTIRCOK	Request completed successfully.
4	4	No	CPTWNEGO	System limits applied to buffer or Queue sizes.
17	11	No	CPTEVRSN	Control block version number not supported.
18	12	No	CPTECONN	Required Parameter not passed. For example, host, port, …
20	14	No	CPTETOKN	Specified data transfer token is invalid.
31	1F	No	CPTEFRMT	Other Socket Call Parameter List format or specification error.
34	22	No	CPTENAPI	API not fully available; retry.
36	24	No	CPTEDRAN	TCP/IP environment is terminating.
40	28	Yes	CPTETERM	Environment is being terminated.
47	2F	Yes	CPTEENVR	Other transport layer environmental condition.
65	41	Yes	CPTERLSE	Orderly release of remote connection request.
68	44	Yes	CPTEDISC	Remote connection not available or aborted.

Decimal	Hex	Diagnostic Code	Variable	Description
72	48	Yes	CPTEPRGE	Remote connection environment terminating.
79	4F	Yes	CPTEINTG	Other transport layer connection/data integrity error.
143	8F	Yes	CPTEPROC	Procedural error.
254	FE	Is abend code	CPTABEND	Abnormal termination. The diagnostic code is the abnormal termination code that is normally a CICS abend code, but can also be in the “Abend Codes” chapter of the *Message Guide*.
255	FF	No	CPTEOTHR	Other error.

Usage Notes

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

There are required and optional fields initialized by a user or calling application. The calling program must identify the server. You specify the server by selecting the remote IP address field name ACMRADDR (or DNS name ACMRNAME) with the remote port ACMRPORT. The remote port selection defines the server's well-known port address. Optional fields control data transfer buffering, statistics, and subtask initialization.

When the CONNECT service completes, the ACM control block contains information related to the established connection. A token ACMTOKEN identifying 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.

WARNING! *Manipulating the token in any way can cause unpredictable results.*

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 two 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 CONNECT 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. Specifying a remote host name in ACMRNAME makes the code more flexible (Since the host IP address could change and the program would not need to be modified.) However, there is extra overhead required to resolve a host name to an IP address with a DNR call.

The remote port number ACMRPORT is a required field. This field identifies the well-known port to which the CONNECT service initiates a connection request.

It is recommended that programmers set the ACMNODNR field to prevent the extra overhead of making DNR calls to resolve the remote IP address whenever possible.

User application programs have the ability to control Unicenter SOLVE:CPT and transport provider data transfer buffering. ACMMSEND and ACMMRECV specify the 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 product of the queue and buffer values cannot exceed 32 KB. CPT requires some additional storage to manage these buffers. This extra storage is **not** included in the allocation.

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. Consider enabling the statistics option to gather CPT statistical information, which can be used to set the SEND or RECEIVE 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 not typically 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.

Complete Parameter List

ACMBCKLG	Maximum size of the LISTEN backlog queue. Not used by the CONNECT service.
ACMCDTBL	Not used by the CONNECT service.
ACMCLNTL	Not used by the CONNECT service.
ACMDGNCD	Indicates the diagnostic code received by the CONNECT service for a transport provider request. The ACMDGNCD depends on the error event recorded in the ACMRTNCD field. The ACMDGNCD could be CICS abend code, ERRNO, or other value depending on the ACMRTNCD failure. When a Unicenter SOLVE:CPT API call fails, the product prefers to return the ACMRTNCD and ACMDGNCD pair from the first error event that occurred during the Unicenter SOLVE:CPT API call. An API system Error return code (ERRNO) can be mapped back into a Unicenter SOLVE:CPT return code (ACMRTNCD) when an EZASOKET (or EZACICAL) error occurs during processing of a Unicenter SOLVE:CPT API call. If the first error on a Unicenter SOLVE:CPT API call is an EZASOKET (or EZACICAL) error, then the ACMDGNCD contains the TCP API system Error return code (ERRNO). To determine the meaning of the ERRNO number, see IBM's *Communication Server IP API Guide* or IBM's *Communication Server IP CICS Sockets Guide* or equivalent.
ACMFUNC	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.
ACMLADDR	Local IP host address. Indicates the local host internet address. The local host internet address is updated when a server connection is established, and is returned to the caller. This field is an unsigned four-byte integer value. Default: None.
ACMLNAME	Local IP host name. Indicates the local host internet name. The local host internet name is updated when a client connection is established, and is returned to the caller. This field is a 255-byte character string that is padded with blanks. Default: None.

ACMLPORT	Client application port. The value returned in this field represents the TCP port on the local host that was assigned to the client application by TCP, if it is not specified by the caller of the CONNECT service. It is a standard practice to not specify a value for this parameter on the CONNECT service call. Not specifying a value allows the transport provider to assign this local port for you. If the caller of the CONNECT service does specify a local port, the call could fail if the port is already used by TCP. This field is an unsigned positive integer with a maximum value of 65,534. Default: None.
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. Total allocation cannot exceed 32 KB. Default: 1024.
ACMMROAS	Not used by the CONNECT service.
ACMMROEP	Not used by the CONNECT service.
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. Total allocation cannot exceed 32 KB. Default: 1024.
ACMMSOCK	Maximum sockets per allowed for your transaction This field overrides the MSOCK= value from the T09MCICS configuration macro. Default: 50.

ACMOPTNS	Specifies TCP connection initialization options.	
	ACMCTRAN	Not used by the CONNECT service.
	ACMLTRAN	Client-Data Listener option. This option is for the LISTEN service and is not validated or modified by the CONNECT service.
	ACMNODNR	DNR Suppression option. Skip internal DNR calls to resolve the requested DNS host name into a remote IP address in the ACMRADDR field. Default: DNR is used for host name resolution.
	ACMOCLEN	Not used by the CONNECT service.
	ACMOMRO	Not used by the CONNECT service.
	ACMOSEC	Not used by the CONNECT service.
	ACMOTRAN	Not used by the CONNECT service.
	ACMOUSR	Not used 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.
	ACMOP2SC	Specifies whether a comma can be part of the first data packet passed to the CSKL replacement listener. It permits one or more commas in the first data packet. Not used by the CONNECT service.
	Default: None.	
ACMOPTN3	Specifies TCP connection initialization options.	
	ACMO3NOT	Specifies that the session will not participate in the STEAR GIVE and session inactivity timeouts.
	ACMO3CPE	Extended CONNB session. Internal product use only.
	ACMO3INC	Internal call. Internal product use only.
	Default: None.	

ACMQRECV	API receive queue size. You should only specify one. Adding extra buffers wastes storage and does not improve performance. Default: One.
ACMQSEND	API send queue size. You should only specify one. Adding extra buffers wastes storage and does not improve performance. Default: One.
ACMRADDR	Remote IP host address. Indicates the remote host internet address. Either this field or the remote host name (ACMRNAME) field must be specified. The remote host internet address is updated when a server connection is established, and is returned to the caller. This field is an unsigned four-byte integer value. Default: None.
ACMRNAME	Remote IP host name. This field 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. It can also be used to resolve a dotted decimal name such as "123.234.123.234." The remote host internet name is updated when a server connection is established, and is returned to the caller. Default: None.
ACMRPORT	Required. Remote well-known service port. This value represents the TCP port on the remote host to which the client application is trying to connect. This field is an unsigned positive integer with a maximum value of 65,534. Default: None.
ACMRTNCD	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. For expected values, see Return Codes. See the ACMDGNCD parameter above.
ACMSECLM	Not used by the CONNECT service.

ACMSRVCE	This field remains only for downward compatibility purposes and is ignored. This field is no longer supported in version 6 of CPT.
ACMSTATS	Specifies statistics logging options for the application program. The facility can be used for debugging and tuning during development.

ACMSCONN	Specifies that messages be generated on establishing either a listen service or a data transfer connection. These messages are generated by the LISTEN and CONNECT services.
ACMSTERM	Specifies that messages be generated on terminating an established connection. These messages are generated by the CONNECT service.

Default: No statistics logging.

ACMTIMEO — Optional. This field is used by the CONNECT service to specify the amount of time in seconds to wait for a TCP connection to complete.

Default: 30.

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.

ACMTOKEN — Specifies the token created and returned by the CONNECT service. This token is used for all subsequent service calls for the client connection. Applications should initialize this field to zero.

ACMTRACE — Note that the tracing functionality has moved in version 6 of Unicenter SOLVE:CPT. A greatly enhanced tracing capability is now available via the TCPEEP tracing command. These tracing fields remain only for downward compatibility purposes and are ignored. See the *Administrator Guide* for more detail.

ACMTNTRY	ACMTTERM	ACMTTPL
ACMTARGS	ACMTPASS	ACMTRLSE
ACMTRECV	ACMTCLSE	ACMTSTOR
ACMTSEND	ACMTTERR	ACMTCLTD

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.

ACMUCNTX	One word of user context. Specifies one arbitrary word of user context to associate with the connection. The information provided is not interpreted by Unicenter SOLVE:CPT, and is saved with other connection information. Default: Zero, no user context.
ACMUSRID	Not used by the CONNECT service.
ACMVERS	Required. Indicates the version number of the Unicenter SOLVE:CPT argument used by the calling program. Must be set to a binary two for this release of Unicenter SOLVE:CPT. Default: None.

Chapter 5

FTP Client Service

Allows the transfer of files from CICS to remote systems using the Internet standard File Transfer Protocol (FTP). These files are either CICS Transient Data Queues or CICS Temporary Storage.

This chapter includes these sections:

- Call Syntax—Sample syntax for the FTP Client Service call
- Recommended AFT Parameters—Lists the parameters normally used and recommended for the FTP Client Service call
- Usage Example—Provides a sample shell of a program using the FTP Client Service call
- Parameter Values Returned in the AFT—Lists the fields that are updated in the AFT control block upon return from the FTP Client Service call
- Assembler —Provides a list and information about the distributed sample Assembler DSECTs that are used by the FTP Client Service call
- Sample Programs—Lists and describes the distributed sample Assembler programs that use the FTP Client Service call along with other service calls
- Completion Information—Describes the expected results at completion of the FTP Client Service call
- Return Codes—Lists the return codes that can apply to the FTP Client Service call
- Module Descriptions —General descriptions of the SOLVE:CPT FTP Client Service modules.
- Usage Notes— Miscellaneous notes on usage of the FTP client.
- Complete Parameter List—Provides a complete list of the parameters and their options for the FTP Client Service call

Call Syntax

```
EXEC CICS LINK
PROGRAM ('T09TCFCM')
COMMAREA (CFCAFT)
LENGTH (=Y(AFTLEN))
END-EXEC
```

Recommended AFT Parameters

The following table lists the recommended parameters to use with the FTP Client Service. These parameters are set within the AFT control block; see Assembler DSECTs for sample information.

For a complete list and detailed description of optional parameters, see the Complete Parameter List.

Parameter	Description
AFTFNAMA	Address of remote file name.
AFTFNAML	Length of remote file name.
AFTFTPTL	Length of FTP reply.
AFTFUNC	File transfer type: AFTAPPE for appending to a file. AFTRENM for renaming a file. AFTRETR to retrieve a file. AFTSTOR for storing a file. AFTSTOU for storing a unique file name.
AFTNBRX	Number of files to transfer.
AFTPASS	Password for remote logon.
AFTQNAME	Queue name.
AFTQTYPE	Queue type: AFTQTD for transient data queue. AFTQTS for temporary storage queue.
AFTRNAMA	Address of remote host name.
AFTRNAML	Length of remote host name.
AFTRTNTA	Address of return text.
AFTRTNTL	Length of return text.

Parameter	Description
AFTFTPTA	Address of FTP reply.
AFTTYPE	Transfer Type: AFTASCII for ASCII translation. AFTEBCDC for EBCDIC translation. AFTIMAGE for binary (no translation). AFTLOCAL for local translation.
AFTUSER	User ID for remote logon.
AFTVERS	Version number.

Usage Example

A sample program, T09PAFTP, is provided in the T09SAMP library. The following is based on that sample.

This example contains the minimum amount of information that must be passed to the FTP Client Service for it to function.

The required fields are:

- Remote host name and length, or IP address. If the host name is used, it must be resolvable by your DNR pointed to by Unicenter SOLVE:CPT
- Valid user ID and password on the remote system
- Name and length of the file to be acted upon
- Name and type of queue for data retrieval—required for all functions except rename
- FTP and Unicenter SOLVE:CPT returned text fields and field length

All other fields can be left at the FTP defaults on the system to which the data is being transferred.

Important! *All fields must be specified in the format and case that the remote system requires. The FTP Client Service does no checking before attempting the remote host connection.*

```
*
*       CICS DSECTS AND WORKING STORAGE SECTION
*
DFHEISTG DSECT ,
*
CFCAFT           T09DAFT MF=,PFX=AFT       FILE TRANSFER ARGUMENT
FTPTX            DS    CL80
CPTTX            DS    CL80
*
*       CONSTANTS
*
HOSTNAME         DC    C'remote host name'             -OR-
IPADDR           DC    A(remote host ip address)  i.e. AL1(127,0,0,1)
USERID           DC    C'remote host userid'
PASSWORD         DC    C'remote host password'
FILENAME         DC    C'remote file name'
CFCQNAME         DC    C'TS or TD queue name'
*
*        ENTRY -  C.I.C.S. INITIALIZATION
*
DFHEIENT
*
*    FILL IN HOST NAME -OR- ADDRESS
*
*
          LA          R05,HOSTNAME              LOAD PTR HOST NAME
          ST          R05,AFTRNAMA              SAVE PTR HOST NAME
          L           R05,=A(L'HOSTNAME)        LOAD LEN OF HOST NAME
          ST          R05,AFTRNAML              SAVE LEN OF HOST NAME
*
          MVC         AFTRADDR,IPADDR           SAVE IP ADDRESS
**     FILL IN USERID AND PASSWORD
*
          MVC         AFTUSER,USERID            USER ID
          MVC         AFTPASS,PASSWORD          PASSWORD
*
*    FILL IN FILE NAME STORAGE
*
          LA          R05,FNAME               LOAD PTR TO FILE NAME
          ST          R05,AFTFNAMA            SAVE PTR TO FILE NAME
          L           R05,=A(L'FNAME)         LOAD LEN OF FILE NAME
          ST          R05,AFTFNAML            SAVE LEN OF FILE NAME
*
          MVC         AFTNBRX,=AL4(1)    # of files to transfer (always set to 1)
*
*    FILL IN STORE QUEUE NAME AND TYPE
*
          MVC         AFTQNAME(8),CFCQNAME    TS/TD Q NAME
          MVC         AFTQTYPE(2),=C'TS'      TEMP STORAGE
*
*    FILL IN FUNCTION (STOR, APPE, STOU, RENM)
*
          MVC   AFTFUNC(4),=C'STOR'          FUNCTION=STORE
*
*    FILL IN RETURN INFORMATION FIELD
*
*
          LA     R05,FTPTX                         TP BUFFER ADDRESS
          ST     R05,AFTFTPTA                      SAVE BUFFER ADDRESS
          L      R05,=A(L'FTPTX)                   TP BUFFER LENGTH
          ST     R05,AFTFTPTL                      SAVE FTP BUFFER LENGTH
          LA     R05,CPTTX                         PT BUFFER ADDRESS
          ST     R05,AFTRTNTA                      SAVE BUFFER ADDRESS
          L      R05,=A(L'CPTTX)                   TP BUFFER LENGTH
          ST     R05,AFTRTNTL                      AVE FTP BUFFER LENGTH
```

```
*-------------------------------------------------------------*
*
*   LINK TO CPT/FTP CALLABLE CLIENT
*
*-------------------------------------------------------------*
          EXEC  CICS LINK                                          X
                  NOHANDLE                                         X
                  PROGRAM         ('T09TCFCM')                     X
                  COMMAREA        (CFCAFT)                         X
                  LENGTH          (=Y(AFTLEN))
*
*   CHECK RESULTS
*
          L       R07,AFTRTNCD                 LOAD CPT RETURN CODE
          L       R08,AFTDGNCD                 LOAD CPT DIAG CODE
          L       R09,AFTFTPCD                 LOAD FTP RETURN CODE
*
*        Processor and record ant errors
*
          EXEC    CICS RETURN
```

Sample Program T09PAFT3

PProgram T09PAFT3 can be found in the ***cpthlq***.T09SAMP data set. It is invoked by transaction IAF3. Sample RDO definitions can be found in the T09RDOSM member in the ***cpthlq***.T09SAMP data set for the program and its transaction.

Program T09PAFT3 must be configured, assembled and linked by a site before the program can be run. The Transient data queues must be configured in the CSD and the CICS startup JCL before a site can run the T09PAFT3 program.

T09PAFT3 is a user application program that invokes the CPT ftp callable client to write a file from TDQUEUE QUEUEIN. It then invokes the cpt ftp callable client to receive the file back into a different TDQUEUE QUEUEOUT.

Modify the following variables in the FTP job T09PAFT3 to get it to work in your CICS region:

```
HOSTNAME DC    C'MVS.COM'           HOST NAME
IPADDR   DC    AL1(141,202,198,145) Host IP ADDR
QUEUEIN  DC    CL4'TEIN'            TD QUEUE input
QUEUEOUT DC    CL4'TEOU'            TD QUEUE OUTPUT
MYUSERID DC    CL8'HLQUSER '
MYPASSWD DC    CL8'PASSWORD'
```

Suppose a site set the variables inside program T09PAFT3:

```
HOSTNAME DC    'USCMCT4P.CA.COM'    HOST NAME
...
IPADDR   DC    AL1(141,202,198,145) Host IP ADDR
QUEUEIN  DC    CL4'TEIN'            TD QUEUE input
QUEUEOUT DC    CL4'TEOU'            TD QUEUE OUTPUT
MYUSERID DC    CL8'CICSUSR '        Userid
MYPASSWD DC    CL8'CICSPSWD'        Password
```

The variables as set above do the following:

- User CICSUSR is signed on to the remote system using password CICSPSWD
- The contents of the TDQUEUE TEIN is transferred to host 141.202.198.145 into a file called file CICSUSR.TDQUEUE.FTPECHO using the CPT FTP callable interface
- File CICSUSR.TDQUEUE.FTPECHO is transferred from IP address 141.202.198.145 into TDQUEUE TEOU.

Tthe following JCL was added to the CICS regions to get it for use by my TEIN and TEOU transient data queues:

```
//*
//*      Extradition data set TDQUEUES TEIN and TEOU
//*
//TDEXTIN  DD  DISP=SHR,DSN=CICSUSR.TDQUEUE.INPUT
//TDEXTOUT DD  DISP=SHR,DSN=CICSUSR.TDQUEUE.OUTPUT
```

These files have the following DCB characteristics:

```
Command - Enter "/" to select action                 Dsorg  Recfm  Lrecl  Blksz
-------------------------------------------------------------------------------
         CICSUSR.TDQUEUE.FTPECHO                      PS     FB       80   3120
         CICSUSR.TDQUEUE.INPUT                        PS     FB       80   3120
         CICSUSR.TDQUEUE.OUTPUT                       PS     VB      132   3120
```

The follow CSD updates were used to define the above extra partition transient data queues, programs and transaction for use in a CICS region:

```
//STEP1     EXEC PGM=DFHCSDUP,REGION=1M,
//         PARM='CSD(READWRITE),PAGESIZE(60),COMPAT'
//STEPLIB DD DISP=SHR,DSN=CICSSYS.CTS220.CICS.SDFHLOAD
//DFHCSD  DD DISP=SHR,DSN=QACICS.REGION2.TS22.DFHCSD
//SYSPRINT DD SYSOUT=*
//OUTDD    DD SYSOUT=*
//SYSABOUT DD SYSOUT=*
//SYSABEND DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSIN DD *
*
 DELETE TDQUEUE(TEIN)     GROUP(T09SAMP)
 DELETE TDQUEUE(TEOU)     GROUP(T09SAMP)
 DELETE PROGRAM(T09PAFT3) GROUP(T09SAMP)
 DELETE TRANS(IAF3)       GROUP(T09SAMP)
*
 DEFINE TDQUEUE(TEIN) TYPE(EXTRA)   DDNAME(TDEXTIN) GROUP(T09SAMP)     *
 DATABUFFERS(1) ERROROPTION(IGNORE)    OPENTIME(DEFERRED)              *
 BLOCKSIZE(3120)        RECORDFORMAT(FIXED)    BLOCKFORMAT(BLOCKED)    *
 TYPEFILE(INPUT)        RECORDSIZE(80)                                 *
                        DISPOSITION(SHR)
*
 DEFINE TDQUEUE(TEOU) TYPE(EXTRA)   DDNAME(TDEXTOUT) GROUP(T09SAMP)    *
 DATABUFFERS(1) ERROROPTION(IGNORE)    OPENTIME(DEFERRED)              *
 BLOCKSIZE(3120)        RECORDFORMAT(VARIABLE) BLOCKFORMAT(BLOCKED)    *
 TYPEFILE(OUTPUT)       RECORDSIZE(132)                                *
                        DISPOSITION(OLD)
*
```

```
 DEFINE PROG(T09PAFT3) LANGUAGE (ASSEMBLER) GROUP (T09SAMP)
 DEFINE TRANS(IAF3) PROGRAM(T09PAFT3) TASKDATALOC (ANY)                *
        DESC (T09 ASSEMBLER SAMPLE FTP (CFCC) APPL)     GROUP(T09SAMP)
/*
//
```

Parameter Values Returned in the AFT

After the FTP client call returns control to your application program, the following fields are propagated with valid completion FTP client information. These updated values are passed back to the application in the AFT control block.

Field Name	Description
AFTDGNCD	Diagnostic code.
AFTFTPTL	Actual length of FTP reply.
AFTNBRXT	Number of files transferred.
AFTRTNCD	Return code.
AFTRTNTL	Actual length of returned text.
AFTFTPCD	FTP return code.

Assembler DSECTs

Sample Assembler DSECTs are provided in the distributed software and are available to you in ***cpthlq***.T09MAC.

Variable field names used in the samples and examples in this guide refer to these DSECTs.

T09DAFT — Assembler DSECT name for the AFT. For detailed information and a sample copy of this DSECT, see AFT Control Block Used by FTP Client Call in the "Control Block Layouts" appendix.

Sample Programs

Sample Assembler source code is available in the distributed software in the ***cpthlq***.T09SAMP library. You should be able to find a sample that matches your programming requirement. For complete details on the function a sample program provides, see the program descriptions in the "Unicenter SOLVE:CPT API Services" chapter and the descriptions of the sample members listed below.

Name	Description
T09PAFTP	FTP Client Application.
T09PAFT3	FTP Client which utilizes transient data queues

Completion Information

Completion of a request to the Unicenter SOLVE:CPT FTP Client Service depends on the specified FTP function. For all functions, a control connection is established to the remote host and Telnet logon once this connection is completed. Any specified transfer attributes are also sent to the remote host. Further processing varies depending on the type of FTP function specified.

For functions that cause a file to be transferred (STOR, STOU, and APPE), a data connection is established to the remote host, and the specified file is transferred using the appropriate FTP service command. For remote file management functions (RENM), no data connection is established. The FTP rename service commands are sent through the control connection. The caller's argument list (AFT) is updated to show return codes as described in Return Codes.

On completion of the requested services, control is returned to the invoking user application, with return codes and text indicating the success or failure of the service execution.

Return Codes

The Unicenter SOLVE:CPT FTP Client Service returns status information in return codes and text fields indicating the results of the execution. This information is returned to the invoking user application in five fields of the AFT (COMMAREA). Primary service level status information is returned in AFTFTPCD. This return code indicates overall success or failure of the Unicenter SOLVE:CPT FTP Client Service.

This table describes the SOLVE:CPT FTP Client Service return codes:

Return	Description
0	The service has successfully executed. The specified file has been transferred to the remote host.
4	A noncritical error was detected in one or more of the parms passed in the AFT. No file transfer was attempted.
8	The remote FTP server returned a reply indicating that an error occurred in the file transmission. The file was not successfully transferred.
12	An error was detected in a SOLVE:CPT service routine that necessitates aborting any file transfer in progress.
16	A critical system error has occurred. File transfer is either not attempted or is aborted if already in progress.

AFTRTNTX of the AFT

A text description of the execution results is also returned in AFTRTNTX of the AFT. This text is formatted for display by the invoking user application, and provides a description of the processing status.

See the appropriate *TCP/IP FTP Message Guide* for a detailed explanation of error messages returned in this field.

AFTFTPTX of the AFT

An additional text description of FTP replies is returned in AFTFTPTX of the AFT. This text is formatted for display by the invoking user application and provides the last reply from the FTP remote server. This field indicates the results of the last FTP service command.

See the Internet standard *Request For Comment (RFC) 959* for a further explanation of FTP replies.

Any of the Unicenter SOLVE:CPT services invoked by Unicenter SOLVE:CPT FTP Client Service return a return code and optional diagnostic code indicating the success or failure of the service call. Detected Unicenter SOLVE:CPT errors are returned to your application in AFT fields AFTRTNCD and AFTDGNCD. The diagnostic code is optional and indicates the transport provider return code. See the Return Codes Cross Reference Table.

The following table lists the return codes that can apply to the FTP Client call.

Decimal	Hex	Diagnostic Code	Variable	Description
0	0	No	CPTIRCOK	Request completed successfully.
18	12	Yes	CPTECONN	Required Parameter not passed. For example, host, port, …
19	13	No	CPTEPROT	Specified protocol not supported.
20	14	No	CPTETOKN	Specified data transfer token is invalid.
21	15	No	CPTEBUFF	Buffer address or length invalid.
22	16	No	CPTECHAR	Translate character set is invalid.
23	17	No	CPTEMODE	Translate mode specification is invalid.
31	1F	No	CPTEFRMT	Other Socket Call Parameter List format or specification error.
34	22	No	CPTENAPI	API not fully available; retry.
40	28	Yes	CPTETERM	Environment is being terminated.
47	2F	Yes	CPTEENVR	Other transport layer environmental condition.
65	41	Yes	CPTERLSE	Orderly release of remote connection request.
68	44	Yes	CPTEDISC	Remote connection not available or aborted.
72	48	Yes	CPTEPRGE	Remote connection environment terminating.
79	4F	Yes	CPTEINTG	Other transport layer connection/data integrity error.
143	8F	Yes	CPTEPROC	Procedural error.

Decimal	Hex	Diagnostic Code	Variable	Description
254	FE	Is abend code	CPTABEND	Abnormal termination. Note that the diagnostic code is the abnormal termination code, which is normally a CICS abend code, but can also be in the "Abend Codes" chapter of the *Message Guide*.
255	FF	No	CPTEOTHR	Other error.

Module Descriptions

This section contains a general description of the SOLVE:CPT FTP Client Service modules.

- T09TCFCM
- T09TCCFDM
- T09TCFRM

T09TCFCM

T09TCFCM is the main entry point for requesting Unicenter SOLVE:CPT FTP services. This module is invoked by your application using an EXEC CICS LINK.

Its primary functions are to:

- Accept and validate parameters passed from your application
- Open a control connection to the remote FTP server
- Process access control parameters (USER, PASS, ACCT)
- Process data transfer parameters for each file to be transferred (PORT, TYPE, STRU, MODE, ALLO, SITE, MKD, CWD)
- Open a data connection port to the remote FTP server for each file to be transferred
- Initiate file transfer for each file to be transferred
- Close control connection to remote FTP server and return to your application

T09TCFDM

T09TCFDM is a major subroutine of SOLVE:CPT FTP Client Service that is invoked by an EXEC CICS START issued by T09TCFCM. This subroutine performs the following two primary functions, depending on a function flag set in the invoking routine:

- LISTEN for and accept the data connection from the remote FTP server
- Format and SEND the file to the remote FTP server over the established data connection

T09TCFRM

T09TCFRM is a major subroutine of SOLVE:CPT FTP Client Service that is invoked by an EXEC CICS START issued by T09TCFCM. This subroutine monitors the control connection and RECEIVEs any replies from the remote FTP server.

Usage Notes

CICS programs call FTP Client Service, passing necessary information for the service to accomplish data transmission to a remote FTP server, then return status information. FTP Client Service processes the parameters, negotiates logon and file characteristics, storage and processing attributes, and transmits the specified data to the remote server using standard FTP protocols. Once the transmission occurs, FTP Client Service returns a return code or error text describing the transmission status to the invoking program.

Your application invokes FTP Client Service using an EXEC CICS LINK to the service. Information is exchanged between your application and FTP Client Service through the CICS COMMAREA. To use this service, your application first creates either temporary storage or a transient data file storage queue on the local host for each file that will be transferred to the remote host. The COMMAREA is used to identify the remote host, user access parameters and, for each file to be transferred, the source location, destination names, transfer parameters and storage function. On return from the FTP Client Service, your application is responsible for queue maintenance and, if indicated by the FTP Client Service return codes, retrying any failed transmissions.

This service calls a variety of existing Unicenter SOLVE:CPT TRUE exits to satisfy the file transmission requirements as follows:

T09CCONN	Establishes a connection to the specified remote host.
T09CCLOS	Closes the specified connection to the remote host.
T09CSEND	Sends data to the remote host via the specified connection.
T09CRECV	Receives data from the remote host via the specified connection.
T09CGIVE	Hands off a connection endpoint to another task.
T09CTAKE	Accepts a connection endpoint from another task.
T09CXLAT	Translates data into the appropriate format for transmission or storage.
T09CLFTP	Listens for and accepts a data connection from the remote host.

For each of these called services, the buffering requirements that are normally tunable by the calling user application are internally tuned by the Unicenter SOLVE:CPT FTP Client Service to default values that are appropriate for FTP file transfers.

Complete Parameter List

For a recommended list of parameters, see Recommended AFT Parameters.

AFTACCT

Optional. Account for remote logon. Indicates the account that Unicenter SOLVE:CPT FTP Client Service uses when performing a Telnet logon to the remote host.

Your application is responsible for ensuring that the account number is specified for remote hosts with this type of file access requirement.

Default: None.

AFTALLO

Storage allocation on remote host. Indicates the number of bytes of storage on the remote host to be allocated for the transferred file.

Note: This feature is not currently supported and the field should contain spaces or binary zeros.

Default: Zero.

AFTDGNCD

Diagnostic code. Indicates the diagnostic code associated with a non-zero Unicenter SOLVE:CPT return code. This value generally indicates a transport provide return code.

Default: Zero.

AFTFNAMA

Address of remote file name. Indicates the storage address where the required remote file name is placed.

This 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. This field is a character string of up to 255 bytes. This name can be the last node of the full directory name or the fully qualified data set name or pathname.

When used with STOR or APPE, this field contains the name under which the transferred file will be stored/appended. When used with RENM, this field contains the name of the Rename From file on the remote host.

Default: None.

AFTFNAML

Length of remote file name. Indicates the length in bytes of the remote file name in the storage area defined by AFTNAMA.

This value must be specified if AFTNAMA is specified.

Default: Zero.

AFTFORM	Transmission format.	
	AFTASACC	Use ASA carriage control transmission format (FUTURE).
	AFTFDFLT	Use the FTP service default format.
	AFTNPRNT	Use nn-print transmission format.
	AFTTELNT	Use Telnet transmission format (FUTURE).
	Default: None (use the FTP service default. Refer to the appropriate FTP manual for more details).	
AFTFTPCD	FTP return code. Contains a return code set by the Unicenter SOLVE:CPT FTP Client Service indicating the success or failure of the service request. Default: Zero.	
AFTFTPTA	Address of FTP reply. Indicates the storage address where the final FTP reply text is placed. This is a contiguous segment of storage accessible to the user task. Unicenter SOLVE:CPT FTP Client Service returns one line of text supplied by FTP, indicating the success or failure of the file transfer. Allow a minimum of 80 bytes or the text may be truncated. Default: None.	
AFTFTPTL	Length of FTP reply. Indicates the length in bytes of the FTP reply area available in the storage defined by AFTFTPTA. Default: Zero.	

AFTFUNC	Required. FTP service command. Indicates the four-byte character field FTP service command that should be used for transferring this file.	
	AFTAPPE	Append the file to the file name specified in AFTFNAMA. If the file does not exist, create it.
	AFTRENM	Rename the file specified in AFTFNAMA to the name specified in AFTRNTOA.
	AFTSTOR	Store the file under the name specified in AFTFNAMA. If the file exists, replace it.
	AFTSTOU	Store the file under a unique name as specified in AFTFNAMA in the default or specified working directory. If a file with that name exists, the FTP server reports the unique name assigned to it.
	Default: None.	
AFTMODE	Transmission mode. Indicates the FTP transmission mode to be used for transfer of this file.	
	AFTBLCK	Use Blocked mode (FUTURE).
	AFTCOMP	Use Compressed mode (FUTURE).
	AFTMDFLT	Use the FTP service default mode.
	AFTSTRM	Use Stream mode.
	Default: None. Use the FTP service default. Refer to the appropriate FTP manual for more details.	
AFTNBRX	Number of files to transfer. Indicates the number of files to be transferred for this invocation of Unicenter SOLVE:CPT FTP Client Service.	
	The number of files transferred for a single call is limited to one and the field is not referenced.	
	Default: Zero.	
AFTNBRXT	Number of files transferred. Indicates the number of files that have been transferred successfully to the remote host.	
	The number of files transferred for a single call is limited to one and the field is not used.	
	Default: Zero.	

AFTPASS	Required. Password for remote logon. Indicates the password that Unicenter SOLVE:CPT FTP Client Service uses when performing a Telnet logon to the remote host. Default: None.
AFTQITEM	Number of Temporary Storage items. Indicates the number of items stored in the storage queue specified in AFTQNAME. This field is not currently used. All records in the named queue are processed. Default: Zero.
AFTQNAME	Required. Queue name. Indicates the name of the storage queue that contains the file to be transferred to the remote host. ■ If AFTQTYPE is Transient Data (AFTQTD), this field must contain a four-byte queue name for which there is an existing DCT entry. ■ For Temporary Storage (AFTQTS), this field can contain up to eight characters that identify a TS queue that exists. Default: None.
AFTQTYPE	Required. Queue type. Indicates the type of storage queue used for the file to be transferred. AFTQTD — Stored on a Transient Data queue. AFTQTS — Stored on a Temporary Storage queue. Default: None.
AFTRNAMA	Address of remote host name. Indicates the storage address where the name of the remote host is placed. This 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. Either this value and its associated length (AFTRNAML) or the remote IP address (AFTRADDR) must be specified. The remote host name is a character string of up to 255 bytes. Default: None.

AFTRNAML	Length of remote host name. Indicates the length in bytes of the remote host name in the storage area defined by AFTRNAMA. When AFTRNAMA has been set then the AFTRNAML field should be between 1 and 255. The AFTRNAML field contains an unsigned four byte integer. Default: Zero.
AFTRNTOA	Address of Rename To file name. Indicates the storage address where the name of the Rename To file is placed. This 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. This field contains a character string of up to 255 bytes representing the new name for an existing file identified in AFTNAMA. The Rename To file can be a fully qualified data set name, a full path name, or the last node of the new file name. Default: None.
AFTRNTOL	Length of Rename To file name. Indicates the length in bytes of the Rename To file name in the storage area defined by AFTRNTOA. This value must be specified if AFTRNTOA is specified. Default: Zero.
AFTRTNCD	Return code. Indicates the return code set by Unicenter SOLVE:CPT services called during the Unicenter SOLVE:CPT FTP Client Service file transfer process. Default: Zero.
AFTRTNTA	Address of return text. Indicates the storage address where text describing the Unicenter SOLVE:CPT return code is placed. This is a contiguous segment of storage accessible to the user task. Unicenter SOLVE:CPT FTP Client Service returns one line of text, indicating the success or failure of the file transfer process. Allow at least 80 bytes or the text may be truncated. Default: None.

AFTRTNTL	Length of return text. Indicates the length in bytes of the Unicenter SOLVE:CPT return text area available in the storage defined by AFTRTNTA. Default: Zero.
AFTSITEA	Address of site parameters. Indicates the storage address where optional FTP SITE parameters are placed. This 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. Your application is responsible for ensuring that any SITE parameters are supported by, and consistent with, the requirements of the remote host FTP server. Default: None.
AFTSITEL	Length of site parameters indicates the length in bytes of the FTP SITE parameters in the storage area defined by AFTSITEA. This value must be specified if AFTSITEA is specified. Default: Zero.
AFTSTRU	Transmission structure. Indicates the FTP transmission structure to be used for transfer of this file. AFTFILE — Use File transmission structure. AFTPAGE — Use Page transmission structure (FUTURE). AFTRECRD — Use Record transmission structure. AFTSDFLT — Use the FTP service default structure. Default: None. Use the FTP service default. Refer to the appropriate FTP manual for more details.
AFTRACE	Note that the tracing functionality has moved in version 6 of Unicenter SOLVE:CPT. A greatly enhanced tracing capability is now available via the TCPEEP tracing command. See the *Administrator Guide* for more detail. These tracing fields remain only for downward compatibility purposes and are ignored. AFTTLVL1 AFTTLVL2 Default: Zero (no trace logging).

AFTRADDR	Remote IP host address. Indicates the remote host internet address. Either this field or the remote host name (AFTRNAMA/AFTRNAML) must be specified.
AFTRLIM	Transmission timing. A halfword binary value that specifies the number of retries to attempt before aborting the FTP connection.
AFTTLIM	Transmission timing. A fullword binary value that specifies the maximum amount of time, in seconds, to wait for data to be received from the remote host. If no data is received in this amount of time, the receive is retried.
AFTTYPE	Transmission type. Indicates the FTP transmission type to be used for transfer of this file.

AFTASCII	Use ASCII transmission type.AFTTDFLT Use the FTP service default type.
AFTEBCDC	Use Compressed mode (FUTURE).
AFTIMAGE	Use EBCDIC transmission type (FUTURE).
AFTLOCAL	Use LOCAL transmission type (FUTURE).

Default: None. Use the FTP service default. Refer to the appropriate FTP manual for more details.

AFTUSER	Required. User ID for remote logon. Indicates the user ID that Unicenter SOLVE:CPT FTP Client Service uses when performing a Telnet logon to the remote host. Default: None.
AFTVERS	Required. Version number. Indicates the version number of the FTP Client Service argument used by the calling program. Must be set to a binary two. AFTVERSN—Specifies version number two. Default: None.

AFTWDIRA

Address of working directory name. Indicates the storage address where the name of a working directory is placed.

This 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.

This field contains a character string of up to 255 bytes representing the path name of an existing directory on the remote host.

Unicenter SOLVE:CPT FTP Client Service generates a Change Working Directory command, and creates the path if it does not exist.

Default: None.

AFTWDIRL

Length of working directory name. Indicates the length in bytes of the working directory in the storage area defined by AFTWDIRA.

Note: This value must be specified if AFTWDIRA is specified.

Default: Zero.

Chapter 6

GIVE Service

This service releases ownership of a connection and associated internal Unicenter SOLVE:CPT resources. You must use the GIVE service call to guarantee proper passing of a connection to another transaction.

To invoke the GIVE service, a user application **must first** build an AFM and then issue a call to the GIVE routine. On completion, a return code is set to indicate the success or failure of the request.

This chapter discusses the following topics:

- Call Syntax—Shows sample syntax for the GIVE service call
- Recommended AFM Parameters—Lists the parameters normally used and recommended for the GIVE service call
- Usage Example—Provides a sample program shell using the GIVE service call
- Parameter Values Returned in the AFM—Lists the fields that are updated in the AFM control block on return from the GIVE service call
- Assembler DSECTs - Lists information about the distributed Assembler DSECTs used by the GIVE service call and is available in *cpthlq*.T09SAMP
- Sample Programs—Sample Assembler programs that use the GIVE service call
- Completion Information—Describes the expected results at completion of the GIVE service call
- Return Codes—Lists the return codes that can apply to the GIVE service call
- Usage Notes—Provides notes about GIVE service call usage and resource cleanup.
- Complete Parameter List—List all of the parameters and options of those parameters for the GIVE service call

Call Syntax

We recommend that a site use the T09MCALL macro to call GIVE:

```
LA    R01,CPTAFM
ST    R01,CPTPARMS
T09MCALL GIVE,PARM=CPTPARMS
```

However, a programmer can call the GIVE interface stub directly:

```
LA    R01,CPTAFM
ST    R01,CPTPARMS
LA    R01,CPTPARMS
L     R15,=V(T09FGIVE)
BALR  R14,R15
```

Recommended AFM Parameters

The following table lists the recommended parameters for use with the GIVE service. These parameters are set within the AFM control block. See Assembler DSECTs for sample information.

For a complete list of optional parameters, see the Complete Parameter List.

Parameter	Description
AFMTOKEN	Required session token specifies which session the current task will relinquish control over.
AFMVERS	Version number should be set to 2. The T09MCALL macro automatically sets the correct version number in the AFM parameter list.

Usage Example

In this example, a subset of actual required statements is shown to emphasize the use of a GIVE call. The AFMTOKEN token is loaded from the ACMTOKEN field to be used by the GIVE service. The return code is checked to determine GIVE service completion status.

Note: The statements needed for the GIVE service appear in **bold**.

```
*        Dsect's

         T09DACM MF=DSECT           Argument for Connection Management
         T09DAFM MF=DSECT          Argument for Translate
*
*        Working storage
DFHEISTG DSECT
GIVEARG  DS    XL(AFMLEN)          Argument for Translate
*
label    DFHEIENT
           ...
*
          L    R9,ACMTOKEN        Load ACM Token
*
          ...
*
*        CPT GIVE
*
GIVE     DS    0H
         USING AFM,GIVEARG
*
         ST    R9,AFMTOKEN         Save connection token
*
         LA    R01,GIVEARG         Point to control block
         ST    R01,CPTPARMS        Store address of control block
         T09MCALL GIVE,PARM=CPTPARMS
         LTR   R15,R15             Test Return Code
         BZ    GIVEOK               Good==>GIVEOK
*                                   Bad...
*        GIVE error from AFM
*
         L     R3,AFMRTNCD          Load GIVE Return Code
         L     R4,AFMDGNCD          Load GIVE Diagnostic Code
          ...
*
GIVEOK   DS    0H
*
         EXEC  CICS RETURN
```

Parameter Values Returned in the AFM

After the GIVE call returns control to your application program, the following fields are propagated with information. These updated values are passed back to the application in the AFM control block.

Parameters	Description
AFMDGNCD	Diagnostic code.
AFMRTNCD	Return code.

Assembler DSECTs

Sample Assembler DSECTs are provided in the distributed software and are available to you in ***cpthlq***.T09MAC. Variable field names contained in the distributed samples and the examples in this guide refer to these DSECTs.

T09DAFM — Assembler DSECT name for the AFM. For detailed information and a sample copy of the Assembler DSECT, see AFM: Argument for Facility Management Used by the GIVE and TAKE Services section in appendix “Control Block Layouts.”

All Assembler constants that apply to AFM calls are imbedded in the AFM DSECT sample.

Sample Programs

Sample Assembler source code is available in the distributed software in the ***cpthlq***.T09SAMP library. You should be able to find a sample that matches your programming requirement. For complete details on the function a sample program provides, see the program descriptions in the “Unicenter SOLVE:CPT API Services” chapter and the descriptions of the sample members listed below.

Name	Description
T09PASV3	TCP Server 3 program is a multi-threaded server using a Listen API call with an independent EXEC CICS START tran.
T09PASV5	TCP Server 5 program is spawned by an inbound connection from the T09MLSTN tool. It utilizes the Select tool to handle RECEIVE calls when there is no available data.

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 zero (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 zero indicate the nature of the failure.

Return Codes

The GIVE service returns a code in registers R15 and R0 indicating the results of the execution. These values are in the AFMRTNCD (R15) and AFMDGNCD (R0).

DSECT T09DRTCD contains equates and descriptions for the possible return codes. T09DRTCD is available in the distributed software in ***cpthlq***.T09MAC. See the appendix “Return Codes” for a sample copy of the T09DRTCD DSECT.

The following table lists the return codes that can apply to the GIVE call.

Decimal	Hex	Diagnostic Code	Variable	Description
0	0	No	CPTIRCOK	Request completed successfully.
17	11	No	CPTEVERN	Control block version number not supported.
20	14	No	CPTETOKN	Specified data transfer token is invalid.
34	22	No	CPTENAPI	API not fully available; retry.
37	25	No	CPTESLCT	SELECT Tool transaction is not running.
40	28	No	CPTETERM	TCPIP is terminating.
254	FE	No	CPTABEND	Abnormal termination.
255	FF	No	CPTEOTHR	Other error.

Usage Notes

The GIVE service releases ownership of a connection from a CICS task. Disassociating resources from a task lets the Unicenter SOLVE:CPT properly manage resources during task termination. This ability to GIVE and TAKE ownership of connections offers you a range of programming options, while still providing Unicenter SOLVE:CPT with resource management capabilities.

The GIVE service requires the application to set the AFM version number and token fields. When a connection is established there are internal Unicenter SOLVE:CPT resources associated with that connection. Unicenter SOLVE:CPT is responsible for proper clean up of those resources on task or transaction termination. These resources include storage allocated by Unicenter SOLVE:CPT, the API, and the transport provider storage.

The GIVE is all about proper resource cleanup. For a Unicenter SOLVE:CPT token (connection) to be properly passed to another transaction, it must first be GIVEn to release ownership. The receiving transaction TAKEs the connection.

Important! *If a transaction does not GIVE the token before it performs an EXEC CICS RETURN then the CICS TRUE end of task exit will clean up all resources including closing down the connections. Therefore, if you have not GIVEn your token the next transaction, it cannot use the connection because it will be gone; already be closed; so a TAKE will fail.*

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 Unicenter SOLVE:CPT task termination exit) if the server application terminates. However, if the data transfer transaction is terminated without issuing an explicit close (Unicenter SOLVE:CPT CLOSE service) an implicit close is scheduled, and resource management is handled by the Unicenter SOLVE:CPT task termination exit.

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

The AFMFUNC function code indicates the Unicenter SOLVE: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.

Complete Parameter List

For a recommended list of parameters, see Recommended AFM Parameters.

AFMCOMMA	Reserved for future use.
AFMCOMML	Reserved for future use.
AFMDGNCD	Diagnostic code. Indicates the diagnostic code received by the GIVE service for a transport provider request.
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.
AFMMSOCK	Unused Parameter.
AFMNTRAN	Transaction ID. AFMNTRAN contains the next transaction to be initiated by the Select tool when the ECB is posted inside the SELECT indicating some kind of data activity on the connection. To enable the use of this field, you must specify the AFMOPT-SEL option. Default: Current transaction ID.
AFMOPCD1	AFMOPSEL Informs the GIVE service to pass this token to the SELECT Tool AFMOPCOM Reserved for future use. Default: None.
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: None.
AFMTOKEN	Required session token specifies which session that the current task will relinquish control over.
AFMVERS	Required version number. Indicates the Unicenter SOLVE:CPT version number of the argument list used by the calling program. Must be set to a binary two for this release of Unicenter SOLVE:CPT. Default: None.

Chapter 7

LISTEN Service

Although every effort was made to make the LISTEN service API call as easy as possible to use, it is always easier to use the Unicenter SOLVE:CPT Listen Tool since it requires no coding on your part. Always check first to see if the Listen Tool meets your needs before using the LISTEN service call. It is extremely rare that the Listen tool will not meet your needs. For information on using the Listen tool, see the “Configuration Reference” chapter of the *Administrator Guide.*

The LISTEN service call provides a server facility that is used by an application program. It establishes a session with the local transport provider, passively listens for connection requests. As new session requests come in, it accepts new connections. When a 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 issue a call to the LISTEN routine. The minimum information required by this service is the version number and the local transport provider port. 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 chapter discusses the following topics:

- Call Syntax—Shows sample syntax for the LISTEN service call
- Recommended ACM Parameters—Lists the parameters normally used and recommended for the LISTEN service call
- Usage Examples—Provides sample shells of programs using the LISTEN service call
- Parameter Values Returned in the ACM—Lists fields that are updated in the ACM control block on return from the LISTEN service call
- Assembler DSECTs—Provides information about the distributed sample Assembler DSECTs that are used by the LISTEN service call
- Sample Programs—Lists and describes the distributed sample Assembler programs that use the LISTEN service call along with other service calls.
- Completion Information—Describes the expected results at the completion of the LISTEN service call

- Return Codes—Lists the return codes that can apply to the LISTEN service call
- Usage Notes—Provides miscellaneous notes about LISTEN service call usage
- Network Considerations—Provides a list of consideration when using the ACM, a common data structure, for both client and server connection initialization
- Complete Parameter List—Provides a complete list of the parameters and their options for the LISTEN service call
- Client-Data Listener Option—Describes how to implement the Client-Data Listener Option

Call Syntax

We recommend that a site use the T09MCALL macro to call LISTEN:

```
LA    R01,CPTACM
ST    R01,CPTPARMS
T09MCALL LISTEN,PARM=CPTPARMS
```

However, a programmer can call the LISTEN interface stub directly:

```
LA    R01,CPTACM
ST    R01,CPTPARMS
LA    R01,CPTPARMS
L     R15,=V(T09FLSTN)
BALR  R14,R15
```

Recommended ACM Parameters

The following table lists the recommended parameters for use with the LISTEN service. These parameters are set within the ACM control block. See Assembler DSECTs for sample information.

For a complete list of optional parameters see the Complete Parameter List.

Field Name	Description
ACMLPORT	Listen well-known service port.
ACMOPTNS	Set ACMNODNR to prevent the overhead of DNR calls to resolve the accepted transactions IP addresses.
ACMTRNID	Listen start transaction ID to start when a new connection is received.
ACMVERS	Version number should be set to 2. The T09MCALL macro automatically sets the correct version number in the ACM parameter list.

Usage Examples

Due to the relative flexibility of the LISTEN service call we provide a number of examples of processes involved with the LISTEN service:

- Recommended Server—This is the most common listener style and can be utilized in most server environments.
- Standard Multithreaded Server—A straightforward multithreaded server passes each session to a daughter task.
- Multithreaded Server—Special Start Transaction Needs—This listener style should be used in cases where local work must occur between the LISTEN and start of the daughter transaction to process the new session.
- Single-Threaded Server—This is a rarely used server style, intentionally limiting connections for performance reasons such as access to a critical database
- Client-Data Listener Option—This listener style is rather unique. Therefore, we refer you to another section for further detailed information
- Sample Daughter Task Taking Ownership of a Session—Sample of how a daughter session takes control of a passed server session

Recommended Server

We recommend that most sites configure the Unicenter SOLVE:CPT Listen tool using the T09MLSTN statement in the T09CON*xx* configuration file for each server they wish to run.

Here a server can listen on port 2345 and pass each connection to transaction SRV3 by configuring the T09MLSTN statement in the T09CON*xx* configuration file as follows:

```
T09MLSTN PORT=2345,TRANSID=SRV3
```

Note: *This listener style makes for the most efficient server program. The server application responds more quickly to new connection requests because it is not involved in the task of data transfer or connection management after the initialization connection.*

The ACM control block is passed to each new SRV3 session that can use the EXEC CICS RETRIEVE command to access the ACM related session information fields. To view an example, see the Sample Daughter Task Taking Ownership of a Session.

Standard Multithreaded Server

This sample simply loops listening for new connections initiating the transaction specified by the ACMTRNID field to process the connection. This server style does only one thing, listen, and hands off connections to the daughter transaction as specified in the ACMTRNID field.

Note: This is accepted as the best design for a server.

This multithreaded server example listens for connections on local port 2345. The server starts a new CICS task SRV3 to process each new daughter session. 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). At the point of an error, the ACMRTNCD is checked to determine LISTEN service request completion status.

Note: The statements relating to the LISTEN service appear in **bold**.

```
*       DSECT's
*
      T09DACM MF=DSECT            Argument for Connection Management
*
*       Working storage
*
DFHEISTG   DSECT
           .
LSTNARG    DS     XL(ACMLEN)        Argument for Connection Management
*
*          Entry
*
label      DFHEIENT
           .
*
LISTEN     DS     0H
           MVC    ACMLPORT,=H'2345'      Set Server well-known port
           MVI    ACMOPTN1,ACMNODNR      Set No DNR of IP addr to names
           MVC    ACMTRNID,=CL4'SRV3'    Set TransID to start to SRV3
           LA     R01,LSTNARG            Point to control block
           ST     R01,CPTPARMS           Store address of control block
           T09MCALL LISTEN,PARM=CPTPARMS
           LTR    R15,R15                Test Return Code
           BZ     LISTEN                 Good, loop to process next connection
           L      R4,ACMRTNCD            Load Return Code
           L      R5,ACMDGNCD            Load Diagnostic Code
           .
           . Process and log Connection Management request error
           .
```

Note: There is no need to write the above program for your server. The above server can be replaced by configuring a T09MLSTN macro statement in the T09CON*xx* configuration file as follows:

```
T09MLSTN PORT=2345,TRANSID=SRV3
```

The ACM control block is passed to each new SRV3 session that can use the EXEC CICS RETRIEVE command to access the ACM related session information fields. To view an example, see Sample Daughter Task Taking Ownership of a Session.

Multithreaded Server—Special Start Transaction Needs

If you do not specify the ACMTRNID field in the CPT-ACM LISTEN parameter list, the call to listen returns control for every new connection. There are cases where an application needs to perform some special work between the LISTEN call and the start of the new daughter task.

This example is a multithreaded server application. The server listens on local port 3456. When control is returned from the T09FLSTN call, it can perform any special work. The token is loaded from the ACMTOKEN. The server then uses the GIVE service to release ownership of the session. It starts the daughter transaction to handle the session.

Note: The statements relating to the LISTEN service appear in **bold**.

```
*      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
          MVC    ACMLPORT,=H'3456'    Set Server well-known port
          MVI    ACMOPTN1,ACMNODNR    Set No DNR of IP addr to names
          LA     R01,LSTNARG          Point to control block
          ST     R01,CPTPARMS         Store address of control block
          T09MCALL LISTEN,PARM=CPTPARMS
          LTR    R15,R15              Test Return Code
          BZ     PASSCONN              Good, Branch to process this connection
          L      R4,ACMRTNCD           Load Return Code
          L      R5,ACMDGNCD           Load Diagnostic Code
           .
           . Process and log Connection Management request error
           .
          B      END                   Termination Transaction
PASSCONN  DS     0H
        ...
       Perform any special work
        ...
*           CPT GIVE Facility Management Service
*
          MVC    AFMTOKEN,ACMTOKEN
          LA     R01,GIVEARG            Point to control block
          ST     R01,CPTPARMS          Store address of control block
          T09MCALL GIVE,PARM=CPTPARMS
          LTR    R15,R15               Test Return Code
          BZ     STARTRAN              Good, then start child transaction
*              Process and log GIVE service error
          B      END                   Termination Transaction
STARTRAN  DS     0H
          EXEC   CICS START TRANSID(SRV3) FROM(CPT-ACM) LENGTH()
          B       LISTEN                Listen for more Client connections
*
```

Here the CPT-ACM field is passed to each new SRV3 session that can use the EXEC CICS RETRIEVE command to access the CPT-ACM related session information. To view an example, see Sample Daughter Task Taking Ownership of a Session.

Single-Threaded Server

This is a rarely used server style that intentionally limits connections to one-at-a-time. This style can be used for performance reasons such as severely limiting access to a critical database, to keep the database from consuming too many resources.

Important! *This sample program is generally not the preferred server model. This single-threaded server model is only suitable for connections of very short time duration.*

The problem is that after returning from the LISTEN service the application blocks additional incoming connection requests. All other pending users must wait for the current connection to finish completely before they can use the service.

This example establishes a server connection, processes data, and closes the connection, before finally going back to check for another connection and more work. 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.

Note: The Assembler statements relating to the LISTEN service appear in **bold**. This enables you to see which statements relate to the LISTEN service call.

```
DFHEISTG  DSECT
           .
LSTNARG   DS     XL(ACMLEN)            Argument for Connection Management
GIVEARG   DS     XL(AFMLEN)            Argument for Facility   Management
*
*         Entry
*
label     DFHEIENT
           .
*
LISTEN    DS     0H
          MVC    ACMLPORT,=H'1234'    Set Server well-known port
          MVI    ACMOPTN1,ACMNODNR    Set No DNR of IP addr to names
          LA     R01,LSTNARG          Point to control block
          ST     R01,CPTPARMS         Store address of control block
          T09MCALL LISTEN,PARM=CPTPARMS
          LTR    R15,R15              Test Return Code
          BZ     GOODCONN              Good, Branch to process this connection
          L      R4,ACMRTNCD           Load Return Code
          L      R5,ACMDGNCD           Load Diagnostic Code
           .
           . Process and log Connection Management request error
           .
          B      END                   Termination Transaction
GOODCONN  DS     0H
*
*           Application and CPT data transfer (SEND/RECEIVE) processing
*
     ...
```

Sample Daughter Task Taking Ownership of a Session

A daughter task can take ownership of the session by using EXEC CICS RETRIEVE to get a copy of the ACPT-AFM. It receives ownership of the task by issuing the TAKE command. Sample program T09PASV2 is a sample daughter task program that could have been started by a server to process a session request.

Here is sample code to retrieve the CPT-ACM and take ownership of the session represented by the ACMTOKEN field. It retrieves a copy of the CPT-ACM. It copies the ACMTOKEN session ID into the AFMTOKEN. Ownership of the session occurs after the T09FTAKE (TAKE service) returns with a zero return code.

Note: The statements relating to taking ownership of the session appear in **bold**.

```
*       DSECT's
*
          T09DACL MF=DSECT             Argument for Close
          T09DACM MF=DSECT            Argument for Connection Management
          T09DADT MF=DSECT             Argument for Data Transfer
          T09DAFM MF=DSECT             Argument for Facility Management
*
*       Working storage
*
DFHEISTG  DSECT
           .
CLOSRG    DS     XL(ACMLEN)            Argument for Close
LSTNARG   DS     XL(ACMLEN)           Argument for Connection Management
DATAARG   DS     XL(ADTLEN)            Argument for Data Transfer
TAKEARG   DS     XL(AFMLEN)            Argument for Facility Management
*
*         Entry
*
label     DFHEIENT
           .
*
           EXEC CICS RETRIEVE
                NOHANDLE
                SET  (R3)
                LENGTH (INLENG)
           END-EXEC
*
*           CPT TAKE Facility Management Service
*
          MVC    AFMTOKEN,ACMTOKEN
          LA     R01,TAKEARG          Point to control block
          ST     R01,CPTPARMS         Store address of control block
          T09MCALL TAKE,PARM=CPTPARMS
          LTR    R15,R15               Test Return Code
          BZ     GOODCONN              Good, then process connection
*              Process and log TAKE service error
          B      END                   Termination Transaction
GOODCONN  DS     0H
```

Parameter Values Returned in the ACM

After the LISTEN call returns control to your application program, the following fields are propagated with valid established connection information. These updated values are passed back to the application in the ACM control block.

Field Name	Description
ACMDGNCD	Diagnostic Code.
ACMLADDR	Local IP Host Address.
ACMLNAME	Local IP Host Name.
ACMMRECV	API receive buffer size.
ACMMSEND	API send buffer size.
ACMQRECV	API receive queue size, set to 1.
ACMQSEND	API send queue size, set to 1.
ACMRADDR	Remote IP Host Address.
ACMRPORT	Client Application Port.
ACMTOKEN	Token—Connection or endpoint.
ACMRTNCD	Return Code.

Assembler DSECTs

Sample Assembler DSECTs are provided and are available to you in the distributed software in ***cpthlq***.T09MAC. Variable field names contained in the distributed samples and the examples in this guide refer to these DSECTs.

The Assembler DSECT name for the ACM is T09DACM. For detailed information and a sample copy of the Assembler DSECT, see the ACM: Argument for Connection Management Used by the CONNECT and LISTEN Services section in appendix “Control Block Layouts.”

Sample Programs

Sample Assembler source code is available in the distributed software in the ***cpthlq***.T09SAMP library. You should be able to find a sample that matches your programming requirement. For complete details on the function a sample program provides, see the program descriptions in the "Unicenter SOLVE:CPT API Services" chapter and the descriptions of the sample members listed below.

Name	Description
T09PASV1	TCP Server 1 program is a single-threaded server using a Listen API call.
T09PASV2	Sample daughter session code to process a new session passed from a server.
T09PASV3	TCP Server 3 program is multi-threaded server using a Listen API call with an independent EXEC CICS START tran.
T09PASV4	TCP Server 4 program is a multi-threaded server using a Listen API call that has CPT internally issuing the EXEC CICS START tran.

Completion Information

Completion of a request to the LISTEN service depends on the arguments passed in the ACMTRNID field in the CPT-ACM parameter list.

Completion Information When the ACMTRNID Field Is Set in the CPT-ACM

When the LISTEN service is initiated with a transaction ID (ACMTRNID contains a CICS transaction), 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 client connection token is derived from an ACMTOKEN field passed in the ACMTOKEN field. The CPT-ACM is passed to the daughter task from the server. 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

When a transaction ID is specified in the ACMTRNID field in the CPT-ACM parameter list, 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 return code ACMRTNCD and diagnostic code ACMDGNCD fields should be interrogated to determine the reason for failure. The general return code and the diagnostic code indicate the nature of the failure. The diagnostic code generally contains a specific code that is generated by the transport provider.

Completion Information When the ACMTRNID Field Is Unspecified in the CPT-ACM

When no transaction ID is specified in the ACMTRNID field in the CPT-ACM parameter list, 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.

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

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 in the ACMRTNCD field indicates a successful establishment of a client connection.

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

The return code ACMRTNCD and diagnostic code ACMDGNCD fields should be interrogated to determine the reason for failure. The general return code and the diagnostic code indicate the nature of the failure. The diagnostic code generally contains a specific code that is generated by the transport provider.

Return Codes

The LISTEN service returns a code in registers R15 and R0 indicating the results of the execution. These values are in the ACMRTNCD (R15) and ACMDGNCD (R0). The diagnostic code typically indicates the transport provider return code.

A sample Assembler DSECT is provided in ***cpthlq***.T09MAC, in member T09DRTCD. It details the variable field names contained in the distributed samples and the examples in this guide. See the appendix “Return Codes” for a sample copy of the T09DRTCD DSECT. A description of the problem causing the associated return code is contained in this DSECT.

This table describes the LISTEN service return codes.

Decimal	Hex	Diagnostic Code	Variable	Description
0	0	No	CPTIRCOK	Request completed successfully.
4	4	No	CPTWNEGO	System limits applied to buffers or queue sizes.
6	6	Yes	CPTWBLCK	Non-blocking call to the LISTEN service.
17	11	No	CPTEVERN	Control block version number not supported.
18	12	Yes	CPTECONN	Required parameter not passed. For example, host, port, …
19	13	No	CPTEPROT	Specified protocol not supported.
26	1A	No	CPTETRID	Unable to start a new task.
31	1F	No	CPTEFRMT	Other socket call parameter list format or specification error.
34	22	No	CPTENAPI	API not fully available; retry.
36	24	No	CPTEDRAN	TCPIP environment is no longer accepting any new endpoints.
40	28	Yes	CPTETERM	Environment is being terminated.
47	2F	Yes	CPTEENVR	Other transport layer environmental condition.

Decimal	Hex	Diagnostic Code	Variable	Description
65	41	Yes	CPTERLSE	Orderly release of remote connection request.
68	44	Yes	CPTEDISC	Remote connection not available or aborted.
72	48	Yes	CPTEPRGE	Remote connection environment terminating.
79	4F	Yes	CPTEINTG	Other transport layer connection/data integrity error.
143	8F	Yes	CPTEPROC	Procedural error.
254	FE	Is Abend code	CPTABEND	Abnormal termination. Note that the diagnostic code is the abnormal termination code, which is normally a CICS abend code, but can also be in the "Abend Codes" chapter of the *Message Guide*.
255	FF	No	CPTEOTHR	Other error.

Usage Notes

The LISTEN service lets user-written application programs implement TCP/IP server facilities. Server applications passively wait, then establish connections with single- or multithread support. The LISTEN service generalized parameter list (ACM) describes the application's communications requirements and information related to established connections. The CPT-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, ACMVERSN, and the local port, ACMLPORT, are required. 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 CPT-ACM.

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

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

The transaction ID field, ACMTRNID, identifies the CICS task to process data for a session. 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. An updated CPT-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. ACMMSEND and ACMMRECV specify the 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 32 KB.

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 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. You should consider enabling the statistics option to gather CPT statistical information. This information can set the SEND or RECEIVE 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.

Note: It is recommended that programmers set the ACMNODNR field to prevent the extra overhead of making DNR calls to resolve the remote IP address whenever possible.

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
ACMLNAME	Local IP host name returned to user application.	Local IP host name returned to user application.
ACMLPORT	Local server or listening transport provider well-known port selected by user application.	Local assigned transport provider port returned to user application.
ACMRADDR	Remote IP host address returned to user application.	Remote IP host address selected or returned to user application.
ACMRNAME	Remote IP host name returned to user application.	Remote IP host name selected or 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.
ACMTIMEO	Client-Data Listener timeout value.	
ACMTLSTN	Listen token returned to user application.	
ACMTRNID	Listen START transaction ID.	

Complete Parameter List

Note: For a recommended list of parameters, see Recommended ACM Parameters.

ACMBCKLG — Maximum size of the LISTEN backlog queue.

This field is used to set the size of the LISTEN queue for pending connection requests and overrides the QLSTN= value from the T09MCICS configuration macro.

ACMCDTBL — Translate table. For the Client/Data Listener, specifies the name of the translate table to use for translating the initial input stream.

Default: None.

ACMCLNTL — Client data length. Specifies the maximum length of data the LISTEN service tries to receive for the initial data stream. This value is useful when the amount of client data being sent for the initial stream is different from the normal length of 50 bytes. This speeds processing by having the LISTEN service be able to continue processing without waiting the full ACMTIMEO value for the initial data.

Default: 50.

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: 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).

ACMLADDR — Local IP host address. Indicates the local host internet address. The local host internet address is updated on establishment of a client connection, and is returned to the caller.

This field is an unsigned four-byte integer value.

Default: None.

ACMLNAME — Local IP host name. Indicates the local host internet name. The local host internet name is updated on establishment of a client connection, and is returned to the caller.

This field is a 255-byte character string that is padded with blanks.

Default: None.

ACMLPORT — Required. 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 listens for connection requests.

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.

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.

Total allocation cannot exceed 32 KB.

Default: 1024.

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.

Total allocation cannot exceed 32 KB.

Default: 1024.

ACMMSOCK — Maximum sockets per allowed for your transaction

This field overrides the MSOCK= value from the T09MCICS configuration macro.

Default: None.

ACMOPTNS — TCP connection initialization options.

ACMCTRAN	For Client-Data Listener, indicates that the input stream is to be translated.
ACMLTRAN	Client-Data Listener option. Specifies that the Listen call will receive the input data stream to determine the transaction ID to start. See Client-Data Listener Option for the required input formats and additional information on using this listener type. This option must be used with ACMTIMEO, and should not be used with ACMTRNID.
ACMNBLKO	Not used by the LISTEN service.
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. *Important! It is strongly recommended that you use NODNR option, since this can create huge 30-second connection establishment delays if your DNS is not correctly configured to resolve IP names into IP addresses. Most DNS servers do not support this feature, and the call takes 30 seconds to time out. Therefore, your listening port could be in a blocked state, allowing no new connection establishment for a period of 30 seconds while waiting for the failed DNS call to time out.*
ACMOCLEN	Indicates the ACMCLNTL field is present and valid.
ACMOMRO	Reserved for CPT/MRO feature.
ACMOSEC	Indicates the ACMSECLM field is present and valid.
ACMOTRAN	For Client-Data Listener, indicates that an optional translate table, named in ACMCDTBL, is to be used in the translation process.
ACMOUSR	Indicates the ACMUSRID field is present and valid.
ACMSYNC	Listen Syncpoint option. Issues a CICS syncpoint before starting any transaction from the LISTEN service.
ACMOP2SC	Specifies whether a comma can be part of the first data packet passed to the CSKL replacement listener. It permits one or more commas in the first data packet.

Default: None.

ACMOPTN3	Specifies TCP connection initialization options.

ACMO3NOT	Specifies that the session will not participate in the STEAR GIVE and session inactivity timeouts.
ACMO3CPE	Extended CONNB session. Internal product use only.
ACMO3INC	Internal call. Internal product use only.

Default: None.

ACMQRECV — API receive queue size. You should only specify one. Adding extra buffers wastes storage and does not improve performance.

Default: One.

ACMQSEND — API send queue size. You should only specify one. Adding extra buffers wastes storage and does not improve performance.

Default: One.

ACMRADDR — Remote IP host address. Indicates the remote host internet address. The remote host internet address is updated on establishment of a client connection, and is returned to the caller.

This field is an unsigned four-byte integer value.

Default: None.

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.

ACMMROAS — Reserved for the CPT/MRO feature.

ACMMROEP — Reserved for the CPT/MRO feature.

ACMRPORT — Remote client port. Indicates the remote transport layer address or port. This value is returned to the caller.

This field is an unsigned positive integer with a maximum value of 65,534.

Default: None.

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: None.

ACMSECLM — Security program. Specifies the security exit program to use when a connection request is processed by this LISTEN service.

Note: The ACMOSEC option must also be specified. See the Security Program section of the chapter “CPT API Services” for more information.

Default: None.

ACMSRVCE — This field remains only for downward compatibility purposes and is ignored. It is no longer supported in version 6 of CPT.

ACMSTATS — Specifies statistics logging options for the application program. The facility can be used for debugging and tuning during development.

ACMSCONN	Specifies that a message 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.
ACMSTERM	Specifies that a message be generated on termination of an established connection. These messages are generated by the CPT CLOSE service.

Default: Zero (no statistics logging).

ACMTIMEO — Client-Data Listener timeout values. Specifies the maximum number of seconds that a Listener can wait to receive the client data stream when the ACMLTRAN option is specified.

Default: 30.

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 section) for all other services.

Default: Zero (token returned).

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: Zero (token returned).
ACMTRACE	Note that the tracing functionality was moved in Version 6 of Unicenter SOLVE:CPT. A greatly enhanced tracing capability is now available using the TCPEEP tracing command. See the *Administrator Guide* for more detail. These tracing fields remain only for downward compatibility purposes and are ignored. Default: Zero (no trace logging).
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. **Note:** This field should not be specified if the ACMLTRAN option and ACMTIMEO value are specified. 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 saved with other connection information. Default: Zero (no user context).
ACMUSRID	User ID. Specifies the user ID that this LISTEN service uses if starting daughter transactions (ACMTRNID or ACMLTRAN specified). This allows the started daughter transactions to inherit the security permissions of the specified user ID. The ACMOUSR option must also be specified. If a security exit is used, then the security exit may change the user ID. Default: None.
ACMVERS	Required. Version. Indicates the version number of the CPT argument used by the calling program. It must be set to a binary two for this release of CPT. Default: None.

Client-Data Listener Option

The Unicenter SOLVE:CPT Client/Data Listener option allows one listening TCP/IP socket port to serve as a multi-function server. This is achieved by passing the CICS/TS transaction name in the initial TCP packet. In this way, a single server can distribute connections to many different applications.

This server is compatible with applications written to use IBM's CICS/TS provided listener CSKL.

Important! *This additional server flexibility does have a performance impact. By having the listener do a receive as part of its processing, the servicing of new connections could be delayed. For this reason, this listen server type is not recommended for high connection volume services.*

In an attempt to avoid many of the inherent performance problems, the client-data listener tool service is broken into two transactions:

- The first transaction handles connection establishment thus blocking the port for a minimal amount of time
- The second phase of the listener, waits for the client data independent of blocking the connection establishment port

To further enhance performance:

- Consider using the CLNTLEN parameter whenever possible
- Start multiple client-data listeners

In this way, any high volume applications can be on their own server port independent of low volume applications. There are no restrictions to the number of client data listeners that can be started. By following these suggestions, any possible performance issue can easily be eliminated.

The design of the Client-Data Listener mimics the format of a standard CICS/TS 3270 terminal data stream. In other words, this is very similar to what you are use to seeing come into a standard CICS/TS terminal interaction on initialization of a terminal transaction. The first four characters of the initial data packet is the transaction name as if you were coming from a real 3270 CICS/TS terminal. Another similarity is that the transaction name can be followed by optional data (parameters) that are passed to the transaction. This is a great listener to have for providing multiple applications with TCP connectivity within one long running server transaction. See the previous performance notes for other considerations.

The client data-listener works in the following manner:

When a connection is received, the phase two listener is started to free up (unblock) the original server listening port.

The phase two listener:

- Does a TCP receive from the network
- Expects one of the following client data formats to be received:

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

Depending on the format of data, the listener determines how the actual spawned application daughter is started. Continue reading for further details on how this works.

Coding a value in the CLNTTIME field greater than zero turns on the client-data listener. There are also options for translating the client data string and changing the translation table if that is desired.

Default: No translation.

TRAN | TDQN	A one- to four-character field followed by an optional comma implying more parameters. The field can contain one of the following: ■ The transaction ID to start ■ A transient data queue (TDQ) name to which the 1 to 35-bytes of optional user data is written—if provided
UUUUUUUUUUUU	1- to 35-bytes of user data passed to the started transaction or written to the transient data queue in the field CLNTDATA.
IC	Specifies that transaction TRAN be started in *HHMMSS*. **Note:** If left blank, startup is immediate.
HHMMSS	Hours, minutes, and seconds for the IC option.
TD	Indicates that the optional client data field CLNTDATA(UUUUUUUUU above) will be written into the transient data queue, TDQN.

Client-Data Option Data Structure

The data structure passed to the invoked program has the following format. This structure is accessed by through a EXEC CICS RETREIVE command in the invoked (spawned daughter) transaction. A sample Assembler DSECT with member name of T09DCSKL is provided and is available to you with the distributed software in the ***cpthlq***.T09MAC library. For greater details, see the Client Data Listener Transaction Start section in appendix “Control Block Layouts.”

```
CSKCPARM DS    0F
CSKTOKEN DS    F     New token - socket ID
CSKLNAME DS    CL8   Listener name
CSKLSUBN DS    CL8   Listener subname
CSKCDATA DS    CL35  Up to 35 bytes of client data
         DS    CL1   C language delimiter
         DS    0F
CSKDMN   DS    H     Family
CSKRPORT DS    H     Remote port
CSKRADDR DS    F     Remote IPADDR
         DS    XL8'00'  Reserved
*
CSKLEN   EQU   *-CSKCPARM
```

Examples

Client/Data Listener with Translation

To invoke the Client/Data Listen Tool and automatically translate the input stream from ASCII to EBCDIC, you must specify the following options in the T09MLSTN parameter:

```
T09MLSTN PORT=2002,CLNTIME=5,CLNTRNS=YES,CLNTTBL=MYTABLE,SOCKCOMP=N
```

In this example, the Listen tool:

- Listens for connections on port 2002
- Waits for up to five seconds for the input stream after establishing a connection
- Translates the input stream using the translation table MYTABLE

Important! *When* one specifies *SOCKCOMP=N on the T09MLSTN macro, it creates a CPT session. The daughter session will have a CPT token passed in the CSKTOKEN field of the retrieved CLNT-PARM control block.*

When one specifies SOCKCOMP=Y on the T09MLSTN macro, it creates an EZASOKET API session. The daughter session will have an EZASOKET socket number passed in the CSKTOKEN field of the retrieved CLNT-PARM control block.

Invoking the Listener with Translation from an Assembler CPT

The option, ACMLTRAN, is used in conjunction with ACMTIMEO. It 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.

To invoke the Client-Data Listener from a Assembler CPT API program, you must specify these options in the ACM:

This Option...	Performs this Function...
ACMTIMEO=*nnnn*	Specifies the maximum time the Listen Service waits for the data stream. (Required).
ACMOPTNS=ACMLTRAN	Triggers the Client/Data Listener option. (Required).
ACMOPTNS=ACMCTRAN	Indicates that the input stream should be translated. (Optional).
ACMOPTNS=ACMOTRAN	Indicates that an optional translation table, named in ACMCDTBL, is to be used in the translation process. (Optional).
ACMCDTBL=*table_name*	Specifies the name of the translation table to use for translating the initial input stream. (Optional).
ACMOPTNS=ACMCLEN	Indicates the ACMCLNTL data length field is specified. (Optional).
ACMCLNTL=*nnn*	Specifies the maximum length of data the LISTEN service will try to receive for the initial data stream. This value is useful when the amount of client data being sent for the initial stream is different from the normal length of 50 bytes. This speeds processing by having the LISTEN service be able to continue processing without waiting the full ACMTIMEO value for the initial data. (Optional).

Example of a Assembler Program, Client-Data Listener

- Listens for connections on port 1984.
- Uses automatic translation of the input stream.
- Set the client data field ACMCLNTL to be four (just for the length of the CICS transaction name) in the input stream. This speeds up connection establishment.
- Set the timeout to five seconds for the amount of time to wait for the client data to arrive on the connection.
- Disable resolving IP addresses into DNS hostnames

Note: The statements relating to the LISTEN service appear in **bold**.

```
*      DSECT
*
      T09DACM MF=DSECT            Argument for Connection Management
*
*      Working storage
*
DFHEISTG  DSECT
           .
LSTNARG   DS     XL(ACMLEN)        Argument for Connection Management
*
* TURN ON CLIENT/DATA LISTENER OPTIONS WITH AUTO TRANSLATE; * LENGTH; AND NO DNR
RESOLUTION OF IP ADDRESSES
*
          MVI   ACMOPTN1,ACMLTRAN+ACMCTRAN+ACMNODNR
          MVC   ACMCLNTL,=AL4(4)          Just read the transaction
          MVI   ACMOPTN2,ACMOCLEN
*
          MVC   ACMLPORT,=AL2(1984)       SELECT THE PORT
*
          LA    R01,LSTNARG               Point to control block
          ST    R01,CPTPARMS              Store address of control block
          T09MCALL LISTEN,PARM=CPTPARMS
```

Chapter

8 RCVFROM Service

The RCVFROM (Receive From) service enables you to develop connectionless client and server applications.

This service call is only for UDP applications.

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 Unicenter SOLVE:CPT control blocks at endpoint creation time. On return from the RCVFROM service, ADTTOKEN contains the value that is 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 Unicenter SOLVE:CPT service calls such as CLOSE, GIVE, TAKE, TRANSLATE are available to UDP applications.

The non-blocking option of the RCVFROM service, 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. The other option of course is to use a timeout value in the ADTTIMEO field to make sure that control is returned to your program within a reasonable amount of time.

This chapter discusses the following topics:

- Call Syntax—Shows sample syntax for the RCVFROM service call
- Recommended ADT Parameters—Lists parameters normally used and recommended for the RCVFROM service call
- Usage Example– Provides a sample program shell for using the RCVFROM service call
- Parameter Values Returned in the ADT—Lists fields that are updated in the ADT control block upon return from the RCVFROM service call
- Assembler DSECTs—Provides the distributed Assembler DSECTs that are used by the RCVFROM service call
- Sample Programs—Lists and describes the distributed sample Assembler programs that use the RCVFROM service call along with other service calls
- Network Considerations—Reviews network-related issues that may influence your environment
- Return Codes—Lists return codes that can apply to the RCVFROM service call
- Complete Parameter List—Provides a complete list of the parameters and their options for the RCVFROM service call

Call Syntax

We recommend that a site use the T09MCALL macro to call RCVFROM:

```
LA    R01,CPTADT
ST    R01,CPTPARMS
T09MCALL RCVFROM,PARM=CPTPARMS
```

However, a programmer can call the RCVFROM interface stub directly:

```
LA    R01,CPTADT
ST    R01,CPTPARMS
LA    R01,CPTPARMS

L     R15,=V(T09FRCFR)
BALR  R14,R15
```

Recommended ADT Parameters

The following table lists the recommended parameters for use with the RCVFROM service. These parameters are set within the ADT control block.

See Assembler DSECTs for sample information.

For a complete list of optional parameters, see Complete Parameter List.

Parameters	Description
ADTBUFFA	User data address.
ADTBUFFL	User data length.
ADTOPCD1	Set to ADTTMRCV for timed receive.
ADTTIMEO	RECEIVE timeout value set to a reasonable timeout for your local network.
ADTTOKEN	Data transfer token set to zero(0) for first time, or copied from previous SENDTO or RCVFROM service call.
ADTVERS	Version should be set to 2.

Usage Example

In this example, a subset of the actual statements required is shown to emphasize the use of a RCVFROM call. This example receives data from a remote host. The token is loaded from the ADT and used by all of the following Unicenter SOLVE:CPT service requests. The return code is checked to determine RCVFROM service completion status.

Note: The statements needed for the RCVFROM service appear in **bold**.

```
*        Dsect's
         T09DADT MF=DSECT         Argument for Transfer
*
*        Working storage
DFHEISTG DSECT
RCVFRARG DS    XL(ADTLEN)         Argument for Transfer
*
CPTSBUFF DS    XL80                Data buffer for Message
CPTIOBUF DS    XL80                Returned Data buffer
*
CPTLENG  DS    H                   LENGTH OF INPUT MESSAGE
*
label    DFHEIENT
*
         ...
*
SENDTO   DS     0H
         USING  ADT,RCVFRARG
*
         MVC   ADTRNAME(15),=C'123.234.105.199' Remote IP ADDR
         MVC   ADTRPORT,=AL2(1980)  SELECT THE PORT
         LA    R04,CPTSBUFF         LOAD SEND BUFFER ADDRESS
         ST    R04,ADTBUFFA         SAVE SOURCE ADDRESS
         LH    R05,CPTLENG          LOAD LENGTH OF SOURCE
         ST    R05,ADTBUFFL         SAVE SOURCE LENGTH
*
         LA    R01,RCVFRARG          Point to control block
         ST    R01,CPTPARMS         Store address of control block
         T09MCALL SENDTO,PARM=CPTPARMS
         LTR   R15,R15              Test Return Code
         BZ    CALLRCFR             Good==>CALLRCFR
*                                   Bad,...SENDTO error
         ...
*
CALLRCFR DS    0H
*
*        RECEIVE the remote response.
*
         LA    R03,30              Timeout on RCVFROM 30 Seconds
         ST    R03,ADTTIMEO
         MVI   ADTOPCD1,ADTTMRCV   Set timed receive option
         LA    R04,CPTIOBUF        LOAD RCVFROM BUFFER ADDRESS
         ST    R04,ADTBUFFA        SAVE RECEIVE ADDRESS
         LA    R06,L'CPTIOBUF      LOAD RECEIVE BUFFER LENGTH
         ST    R06,ADTBUFFL        SAVE RECEIVE BUFFER LENGTH
*
         LA    R01,RCVFRARG         Point to control block
         ST    R01,CPTPARMS        Store address of control block
         T09MCALL RCVFROM,PARM=CPTPARMS T09CRCFR ENTRY POINT
         LTR   R15,R15             Test Return Code
         BZ    TRANS                Good==>TRANS
*                                   Bad,...RCVFROM error
         L     R3,ADTRTNCD          Load ADT Return Code
```

```
L       R4,ADTDGNCD          Load ADT Diagnostic Code
```

Parameter Values Returned in the ADT

After the RCVFROM service call returns control to your application program, the following fields are propagated with valid information. These updated values are passed back to the application in the ADT control block.

Parameters	Description
ADTBUFFA	Data buffer filled with data from the network.
ADTBUFFL	The number of user data bytes actually received.
ADTDGNCD	Diagnostic Code.
ADTLADDR	Local IP Host Address.
ADTLNAME	Local IP Host Name.
ADTMRECV	API receive buffer size.
ADTMSEND	API send buffer size.
ADTQRECV	API receive queue size, set to one.
ADTQSEND	API send queue size, set to one.
ADTRADDR	Remote IP Host Address.
ADTRNAME	Remote IP Host Name.
ADTRTNCD	Return Code.
ADTTOKEN	Token—Connection or endpoint.

Assembler DSECTs

Sample Assembler DSECTs are provided in the distributed software and are available to you in ***cpthlq***.T09MAC. Variable field names contained in the distributed samples and the examples in this guide refer to these DSECTs.

T09DADT	Assembler DSECT name for the ADT. For detailed information and a sample copy of the Assembler DSECT, see the ADT: Argument for Data Transfer Used by RECIEVE, SEND, RECVFROM, and SENDTO Services Service section in appendix “Control Block Layouts.”

All Assembler constants that apply to ADT calls are imbedded in the ADT DSECT sample.

Sample Programs

Sample Assembler source code is provided for your use. You should be able to find a sample that matches your programming requirement. For more complete details on what function a sample program provides, read the comments program descriptions in the “Unicenter SOLVE:CPT API Services” chapter and the at the beginning of the sample members listed below. These sample program members are available in the distributed software in the ***cpthlq***.T09SAMP library.

Name	Description
T09PACLU	Sample UDP client.
T09PASVU	Sample UDP server.

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.

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.
ADTLADDR	Local IP host address returned to user application.	Local IP host address returned to 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. This is **not** the recommended setting.	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.

Return Codes

The RCVFROM service returns a code in registers R15 and R0 indicating the results of the execution. These values are in the ADTRTNCD (R15) and ADTDGNCD (R0).

DSECT T09DRTCD contains equates and descriptions for the possible return codes. T09DRTCD is available in the distributed software in ***cpthlq***.T09MAC. See the appendix “Return Codes” for a sample copy of the T09DRTCD DSECT.

The following table lists the return codes that can apply to the RCVFROM call.

Decimal	Hex	Diagnostic Code	Variable	Description
0	0	No	CPTIRCOK	Request completed successfully.
1	1	No	CPTWTIMEO	Timed receive call timed out.
6	6	Yes	CPTWBLCK	Non-blocking call to the RCVFROM service.
17	11	No	CPTEVRSN	Control block version number not supported.
18	12	Yes	CPTECONN	Required Parameter not passed. For example, host, port, …
19	13	No	CPTEPROT	Specified protocol not supported.
20	14	No	CPTETOKN	Specified data transfer token is invalid.
21	15	No	CPTEBUFF	Buffer address or length invalid.
31	1F	No	CPTEFRMT	Other Socket Call Parameter List format or specification error.
34	22	No	CPTENAPI	API not fully available; retry.
40	28	Yes	CPTETERM	Environment is being terminated.
47	2F	Yes	CPTEENVR	Other transport layer environmental condition.
65	41	Yes	CPTERLSE	Orderly release of remote connection request.
68	44	Yes	CPTEDISC	Remote connection not available or aborted.

Decimal	Hex	Diagnostic Code	Variable	Description
72	48	Yes	CPTEPRGE	Remote connection environment terminating.
79	4F	Yes	CPTEINTG	Other transport layer connection/data integrity error.
143	8F	Yes	CPTEPROC	Procedural error.
254	FE	Is abend code	CPTABEND	Abnormal termination. The diagnostic code is the abnormal termination code, which is normally a CICS abend code, but can also be in the "Abend Codes" chapter of the *Message Guide*.
255	FF	No	CPTEOTHR	Other error.

Complete Parameter List

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 Unicenter SOLVE:CPT or the transport provider.

The storage area can be aligned on any boundary convenient for the application program.

Default: None.

ADTBUFFL — Specifies the length in bytes of the ADTBUFFA field when the RCVFROM is issued. After the RCVFROM call completes, ADTBUFFL indicates the actual length returned in ADTBUFFA.

If the incoming datagram does not fit into ADTBUFFA for a length of ADTBUFFL, then the warning, CPTWNEOM is passed back to the caller in ADTRTNCD, indicating that more RCVFROM calls are required to get the entire datagram.

It is an error to call the RCVFROM service with an ADTBUFFL of zero.

ADTDGNCD — Diagnostic code. Indicates the diagnostics code set by the RCVFROM service. This value generally indicates a transport provider return code.

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.

ADTLADDR — Local IP host address. Represents the IP address of the local host and is filled in on return to the client application.

Default: None.

ADTLNAME — Local IP host name

Indicates the local host internet name. The local host internet name is returned to the caller of the RCVFROM service.

This field is a 255-byte character string that is padded with blanks.

Default: None

ADTLPORT

Local well-known service port (used when ADTTOKEN=0). Indicates the local transport layer port on which the calling application will be receiving (RCVFROM) datagrams.

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 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.

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.

Total allocation cannot exceed 32 KB.

Default: 1024

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.

Total allocation cannot exceed 32 KB.

Default: 1024.

ADTMSOCK

Maximum number of sockets per INITAPI. Overrides the MSOCK= value from the T09MCICS configuration macro.

Default: 50

ADTNSLCT

Number of entries in the select vector. Not used by the RCVFROM service.

ADTOPCD1

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.

Important! *It is strongly recommended that you* ***not*** *use the ADTFDNR option, since this can create huge 30 seconds delay in data reception if your DNS is not correctly configured to resolve IP names into IP addresses. Most DNS servers do not support this feature, and the call takes 30 seconds to time out.*

ADTNBLKR

Do not block a call to the RCVFROM service. If no datagrams are currently available, the return code, CPTWBLCK, is returned in ADTRTNCD.

ADTTMRCV

Allows the caller to wait up to a specified amount of time for a datagram. It must be used with the ADTNBLKR option, and ADTTIMEO must be specified.

These options can be toggled on every UDP data transfer call even if the caller is using the same token.

Default: None.

ADTQRECV

API receive queue size. You should only specify one. Adding extra buffers wastes storage and does not improve performance.

Default: 1.

ADTQSEND

API send queue size. You should only specify one. Adding extra buffers wastes storage and does not improve performance.

Default: 1.

ADTRADDR

Remote IP host address. Indicates the remote host IP 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.

ADTRNAME	Remote IP host name. Indicates the remote host internet name. 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. This field is a 255-byte character string that is padded with blanks. 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.
ADTRTNCD	Return code. Indicates the return code set 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
ADTSLCTD	Number of tokens selected. Not used by the RCVFROM service.
ADTSRVCE	This field is only for downward compatibility purposes and is ignored. This field is no longer supported in Version 6 of CPT.

ADTSTAT	Specifies logging options for the application program.	
	ADTSCONN	Specifies that messages be generated on the closing of a UDP token. These messages are generated by the Unicenter SOLVE:CPT CLOSE service.
	ADTSTERM	Specifies that messages be generated on terminating an established connection. These messages are generated by the Unicenter SOLVE:CPT CLOSE service.
	Default: None, no statistics logging.	

ADTTIMEO

RCVFROM time out value.

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: None.

ADTTRACE

Note that the tracing functionality has moved in Version 6 of Unicenter SOLVE:CPT A greatly enhanced tracing capability is now available using the TCPEEP tracing command. See the *Administrator Guide* for more detail.

These tracing fields remain only for downward compatibility purposes and are ignored:

ADTTNTRY	ADTTTERM	ADTTTPL
ADTTARGS	ADTTPASS	ADTTRLSE
ADTTRECV	ADTTCLSE	ADTTSTOR
ADTTSEND	ADTTTERR	ADTTCLTD

ADTUCNTX	One word of user context. Specifies one arbitrary word of user context to associate with the endpoint. The information provided is not interpreted by Unicenter SOLVE:CPT, and is saved with other endpoint information. Default: Zero, no user context.
ADTVECTR	Address of the select vector. Not used by the RCVFROM service.
ADTVERS	Required. Version. Indicates the Unicenter SOLVE:CPT version number of the argument used by the calling program. Must be set to binary two for this release of Unicenter SOLVE:CPT. Default: None.

Chapter 9

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 chapter discusses the following topics:

- Call Syntax—Shows sample syntax for the RECEIVE service call
- Receive Methodology Options—Explains the various methods of architecting your receiving of data, and which ADT options are needed to perform the type of receive logic
- Parameter Values Returned in the ADT—Lists the fields that are updated in the ADT control block upon return from the RECEIVE service call
- Assembler DSECTs—Provides a list and information about the distributed sample Assembler DSECTs that are used by the RECEIVE service call.
- Sample Programs—Lists and describes the distributed sample Assembler programs that use the RECEIVE call along with other service calls
- Completion Information—Describes the expected results at completion of the RECEIVE service call
- Return Codes—Lists the return codes that can apply to the RECEIVE service call
- Usage Notes—Provides miscellaneous notes about usage of the RECEIVE service call
- Complete Parameter List—Provides a complete list of the parameters and their options for the RECEIVE service call

Call Syntax

We recommend that a site use the T09MCALL macro to call LISTEN:

```
LA    R01,CPTADT
ST    R01,CPTPARMS
T09MCALL RECEIVE,PARM=CPTPARMS
```

However, a programmer can call the RECEIVE interface stub directly:

```
LA    R01,CPTADT
ST    R01,CPTPARMS
LA    R01,CPTPARMS
L     R15,=V(T09FRECV)
BALR  R14,R15
```

Receive Methodology Options

Application design drives the selection of a receive methodology. Once you determine the type, refer to the proper receive methodology type section to find the recommended parameters and a usage example for each type.

In TCP communications, data is passed in the form of stream data. This data format is very similar to what the name suggests—it is a stream of data. There is no logical or physical break in the data for records.

Since stream data format is different than the standard record processing used in most MVS style processing, Unicenter SOLVE:CPT provides a wealth of flexibility for easily converting stream data into MVS logical records. In order to use this functionality, you must code options to tell Unicenter SOLVE:CPT how to assemble the records for you.

Important! *By default, none of this record formatting functionality is enabled. You must turn it on with options. Otherwise, you will receive a stream of data as it is sent from the remote.*

Terminology and Receive Concepts Used in the Definitions

The following information describes the concepts and terminology used in RECEIVE service processing.

BLOCKING

Blocking means that the RECEIVE call can wait until the expected data is received over the ADTTIMEO timeout interval.

From a CICS perspective, a blocking RECEIVE call creates a long running CICS task. Long running CICS tasks are shunned as a poor programming practice.

NON-BLOCKING

Non-blocking does not wait on a RECEIVE call. The RECEIVE checks for data and returns back to the caller either with the data or with a CPTWBLCK return code stating that there is no data available at this time

The preferred non-blocking method involves integrating RECEIVE calls with the SELECT tool.

SELECT Tool

The SELECT tool can monitor outstanding RECEIVE calls for many CICS transactions. When the RECEIVE data is available the SELECT tool fires off a transaction that can issue a RECEIVE call to RECEIVE data for the session.

A SELECT tool RECEIVE can be combined with LL, separator character, or timed RECEIVE methods.

LL RECEIVE Option

When a caller specifies the LL receive option type, it expects the data stream to contain a two-byte length field (LL) followed by data bytes. A binary length contained in the first two positions of the received data stream determines the length of the expected data record.

LL RECEIVE Example

In the following hexadecimal example of a data stream, the two-byte LL header is in **bold**:

0008E3C5E2E3D9C5C3F1**000A**D4E8E3C5E2E3D9C5C3F2

Length	Data Record 1	Length	Data Record 2
0008	E3C5E2E3D9C5C3F1	000A	D4E8E3C5E2E3D9C5C3F2

where:

0008	LL header value denotes eight bytes of data to follow the LL characters.
TESTREC1	Actual character data in the first record.
000A	LL header value denotes ten bytes of data to follow the LL characters.
MYTESTREC2	Actual character data in the second record.

Separator Character RECEIVE Option

In the separator character receive option type, the end of a record is determined by finding one or two separator characters.

Note: The data may never contain a natural occurrence of the separator characters and is a major limitation of the separator character RECEIVE option.

Double Separator Character Example

In the following hexadecimal example, data stream, uses two separator characters: CRLF (carriage return) shown in **bold**:

8E3C5E2E3D9C5C3F1**0D0A**D4E8E3C5E2E3D9C5C3F2**0D0A**

Data Record 1	Separator Characters	Data Record 2	Separator Characters
E3C5E2E3D9C5C3F1	0D0A	D4E8E3C5E2E3D9C5C3F2	0D0A

where:

TESTREC1	Data in the first record.
0D0A	CRLF separates the first from the second data record.
MYTESTREC2	Data in the second record.
0D0A	CRLF separates the second and any following records.

Single Separator Character Example

In the following hexadecimal example data stream, using x'FF' as a separator character shown in **bold**:

8E3C5E2E3D9C5C3F1**FF**D4E8E3C5E2E3D9C5C3F2**FF**

Data Record 1	Separator Characters	Data Record 2	Separator Characters
E3C5E2E3D9C5C3F1	FF	D4E8E3C5E2E3D9C5C3F2	FF

where:

TESTREC1	Data in the first record.
0D0A	FF separates the first from the second data record.
MYTESTREC2	Data in the second record.
FF	Separates the second and any following records.

Timed RECEIVE

When a CPT application uses timed RECEIVE calls, it is up to the programmer to figure out when they have received all their data. The application may have to issue multiple RECEIVE calls as data may be broken into multiple packets.

Once the RECEIVE call returns it is the programmer's responsibility to analyze the return codes and ADTBUFFL data length to determine what to do next. This may include saving any partial packet that was received into the user's data area.

In a partial time RECEIVE call, an endpoint waits until any data is received over the timeout interval ADTTIMEO.

In a full timeout RECEIVE call, an endpoint waits over the timeout interval of ADTTIMEO for all the data (as specified by ADTBUFFL) to arrive from the network.

Introduction to Receive Methodology Options

The following table identifies most of the receive methodologies that can occur. Review this table to determine which method best fits your design. Then follow the link to the section to examine recommended ADT options and samples.

Non-Blocking Fixed Length	The length of the expected data is known. When the RECEIVE is issued with the no wait option, one of the following should occur under normal circumstances: ■ The proper amount of data is available and returned to the caller. ■ No data is returned with a will block (CPTWBLCK) reason code (6) set in the ADTRTNCD field of the ADT. When the wait condition is received, the caller gives the token over to the SELECT tool, which wakes the application up when the requested data is available.
Non-Blocking Variable Length RECEIVE	The length of the expected data is unknown. The RECEIVE is issued with the ADTNWAIT no wait option along with a timeout interval ADTTIMEO. One of the following should occur under normal circumstances: ■ The data is available and returned to the caller ■ No data is returned with a will block (CPTWBLCK) reason code (6) set in the ADTRTNCD field of the ADT. When the wait condition is received, the caller gives the token over to the SELECT tool, which wakes the application up when the requested data is available.

Non-Blocking LL

A binary length contained in the first two characters of the received data stream determines the length of the expected data record. When the RECEIVE call is issued with a no wait option; one of the following should occur under normal circumstances:

- The proper amount of data is available and returned to the caller.
- No data is returned with a will block (CPTWBLCK) reason code (6) set in the ADTRTNCD field of the ADT. When the wait condition is received, the caller gives the token over to the SELECT tool, which wakes the application up when more data is available.

Non-Blocking Separator Character

The records are determined based on finding one or two separator characters. See the example in the definitions section. When the RECEIVE call is issued with a no wait option; one of the following should occur under normal circumstances:

- The proper separator characters delimited record is available and returned to the caller.
- No data is returned with a will block (CPTWBLCK) reason code (6) set in the ADTRTNCD field of the ADT. When the wait condition is received, the caller gives the token over to the SELECT tool, which wakes the application up when more data is available.

Blocking Fixed Length

The length of the expected data is known. The receive waits for all data to be received or until the ADTTIMEO timeout expires.

Blocking LL

A binary length contained in the first two characters of the received data stream determines the length of the expected data record. This length is then used to issue a receive that waits until all data is received or an ADTTIMEO timeout expires.

Blocking Separator Character

The records are determined based on finding one or two separator characters. The data is received until the separator characters are found or an ADTTIMEO timeout expires. Then, the record is given to the RECEIVE caller.

Non-Blocking RECEIVE

With this method, the application continuously issues no wait receives holding the data and looping back to receive more data until the data is exhausted or the application determines that it has what it needs.

Blocking RECEIVE

This is the TCP sockets default of continuously issuing receives, holding the data and looping back to receive more data until the data is exhausted or the application determines that it has what it needs.

What makes this a blocking receive call is that the call must receive some data (or a failure at the transport provider) or control will never be returned to the caller. This is why in the recommendation below we suggest a timeout.

Any of the previous methodologies have valid uses and we can recommend them. However, we do **not** recommend issuing a blocking receive call without a timeout.

Important! *You should always set a timeout in the ADTTIMEO field whenever you issue a blocking receive. Even a non-blocking receive should eventually hit a timeout.*

Non-Blocking Fixed Length RECEIVE

The length of the expected data is known. The RECEIVE is issued with a no wait option. Either the proper amount of data is available and returned to the caller; or no data is returned with a will block (CPTWBLCK) reason code (6) being set in the ADTRTNCD field of the ADT. If a wait condition is received, the caller gives the token over to the SELECT tool, which wakes the application up when the requested data is available.

Recommended ADT Parameters

The following table lists the recommended parameters to use with the Non-Blocking Fixed Length RECEIVE methodology. These parameters are set within the ADT control block; see Assembler DSECTs for sample information.

For a complete list of optional parameters, see Complete Parameter List.

Parameters	Description
ADTBUFFA	Set to address of user data area.
ADTBUFFL	Set to expected fixed length record.
ADTOPCD1	Receiving method: set to ADTTMRCV.
ADTOPCD2	Receiving method: set to ADTNWAIT.
ADTTIMEO	RECEIVE timeout value, set to reasonable timeout for your network environment. Issuing RECEIVE calls with the SELECT tool does not create a long running transaction. So, it is okay to set the ADTTIMEO to a value such as one minute.
ADTTOKEN	Data transfer token.
ADTVERS	Version number should be set to 2. The T09MCALL macro automatically sets the correct version number in the ADT parameter list.

Sample Program Usage

This partial program shows a method of non-blocking fixed length receiving. A connection is made and the token is loaded from the ACM and used by all of the following Unicenter SOLVE:CPT service requests. The length of the expected data is known, and 345 is moved into the ADTBUFFL field. ADTOPTNS is set to ADTNWAIT and ADTTMRCV for full block receiving without waiting. Control returns to the program immediately.

The programmer can determine when data is received by checking both the updated length in the ADTBUFFL field and the ADTRTNCD return code completion status.

A return code of CPTWTIMO (1) in the ADTRTNCD field indicates the RECEIVE has timed out in the SELECT tool waiting for data.

The programmer should check the ADTRTNCD for CPTWBLCK (6), which indicates that the RECEIVE call will block. In the CPTWBLCK case, the program hands the connect token over to the SELECT tool by:

- Setting option AFMOPSEL
- Moving the transaction to be kicked off into field AFMNTRAN
- Calling the GIVE service as shown at label GIVESLCT

When transaction NXTR gets control again from the SELECT tool, the RECEIVE call must be issued with the exact same options set, otherwise the results are unpredictable.

Note: Relevant parameters of the example are in **bold**.

```
*        Dsect's
         T09DACM MF=DSECT          Argument for Connection Management
         T09DADT MF=DSECT         Argument for RECV
         T09DAFM MF=DSECT          Argument for GIVE/TAKE
*
*        Working storage
DFHEISTG DSECT
CPTPARMS DS    F                   CPT calling paramater
RECVARG  DS    XL(ADTLEN)         Argument for RECV
TAKEARG  DS    XL(AFMLEN)          Argument for GIVE/TAKE
CPTIOBUF DS    XL1024              CPT RECV/SEND BUFFER
ARGLENG  DS    H
*
label    DFHEIENT
*
         ...
*
     EXEC CICS RETRIEVE
          NOHANDLE
          SET  (R09)
          LENGTH (ARGLENG)
     END-EXEC
*
         CLC   ARGLENG,=AL2(ACMLEN)   HAVE WE GOT AN ACM BLOCK?
         BE    FIRSTTAK                   NO, PSEUDO CONVERSATIONAL
*                                             REENTRY WITH ONLY TOKEN
*
         L     R09,0(,R09)         GET TOKEN FROM COMMAREA
*                                  PASSED FROM SELECT TOOL
         B     COMMNTAK            JOIN COMMON TAKE LOGIC
*
*        TAKE THE CONNECTION FROM THE LISTENER
*
FIRSTTAK DS    0H
         L     R09,ACMTOKEN-CPTACM(,R09) GET TOKEN FROM COMMAREA ACM
*
COMMNTAK DS    0H
         USING AFM,TAKEARG
         ST    R09,AFMTOKEN        ASSOCIATE FACILITY MANAGEMENT
         ST    R09,ADTTOKEN        ASSOCIATE DATA TRANSFER

*
         LA    R01,TAKEARG         Point to control block
         ST    R01,CPTPARMS        Store address of control block
         T09MCALL TAKE,PARM=CPTPARMS
         LTR   R15,R15             Test Return Code
         BZ    TAKEOK              Good==>TAKEOK
*                                  Bad...
*        TAKE error from AFM
*
         L     R3,AFMRTNCD         Load GIVETAKE Return Code
         L     R4,AFMDGNCD         Load GIVETAKE Diagnostic Code
          ...
*
TAKEOK   DS    0H
RECV     DS    0H
         USING ADT,RECVARG
*
         ST    R9,ADTTOKEN         Save connection token
         LA    R04,CPTIOBUF         LOAD I/O BUFFER ADDRESS
         ST    R04,ADTBUFFA        SAVE RECEIVE BUFFER ADDRESS
         LA    R05,345              LOAD known data length
```

```
         ST    R05,ADTBUFFL        SAVE RECEIVE BUFFER LENGTH
         LA    R03,60               LOAD Receive Timeout
         ST    R03,ADTTIMEO        SAVE RECEIVE Timeout
         MVI   ADTOPCD2,ADTNWAIT   No-Wait Use Select TOOL
         MVI   ADTOPCD1,ADTTMRCV   Timed Receive
*
         LA    R01,RECVARG         Point to control block
         ST    R01,CPTPARMS        Store address of control block
         T09MCALL RECV,PARM=CPTPARMS
         LTR   R15,R15             Test Return Code
         BZ    RECVOK               Good==>RECVOK Have data
*                                   What happened?
         CLC   ADTRTNCD,=AL4(CPTWBLCK)     Receive block?
         BE    GIVESLCT                    Yes==>Give to SELECT TOOL
         CLC   ADTRTNCD,=AL4(CPTERLSE)     No more data?
         BE    CONNCLOS                    Yes==>Connection CLOSEing
         CLC   ADTRTNCD,=AL4(CPTWTIMO)     Timeout occur?
         BE    RECV                        Yes==>RECV
*                                          Bad...
*        RECV error from ADT
*
         L     R3,ADTRTNCD          Load RECV Return Code
         L     R4,ADTDGNCD          Load RECV Diagnostic Code
          ...
*
RECVOK   DS    0H
          ...
*
GIVESLCT DS    0H
         USING AFM,TAKEARG
*
         ST    R9,AFMTOKEN         Save connection token
         MVC   AFMNTRAN,=C'NXTR'   NEXT TRANSACTION ID TO START
         MVI   AFMOPCD1,AFMOPSEL   GIVE TO SELECT TOOL
*
         LA    R01,TAKEARG         Point to control block
         ST    R01,CPTPARMS        Store address of control block
         T09MCALL GIVE,PARM=CPTPARMS
         LTR   R15,R15             Test Return Code
         BZ    GIVEOK               Good==>GIVEOK
*                                   Bad...
*        GIVE error from AFM
*
         L     R3,AFMRTNCD          Load GIVE Return Code
         L     R4,AFMDGNCD          Load GIVE Diagnostic Code
          ...
*
GIVEOK   DS    0H
*
         EXEC  CICS RETURN
```

Non-Blocking Variable Length RECEIVE

The length of the expected data is unknown. The RECEIVE is issued with the ADTNWAIT no wait option along with a timeout interval ADTTIMEO. One of the following should occur under normal circumstances:

- The data is available and returned to the caller
- No data is returned with a will block (CPTWBLCK) reason code (6) set in the ADTRTNCD field of the ADT. When the wait condition is received, the caller gives the token over to the SELECT tool, which wakes the application up when the requested data is available.

Recommended ADT Parameters

The following table liststhe recommended parameters for use by the Non-Blocking Variable Length RECEIVE methodology. These parameters are set within the ADT control block; see Assembler DSECTs for sample information.

For a complete list of optional parameters, see Complete Parameter List.

Parameter	Description
ADTBUFFA	Set to address of user data area.
ADTBUFFL	Set to maximum buffer size in the ADTBUFFA field.
ADTOPCD1	Receiving method: set to ADTTMPRT.
ADTOPCD2	Receiving method: set to ADTNWAIT.
ADTTIMEO	RECEIVE timeout value, set to reasonable timeout for your network environment. Issuing RECEIVE calls with the SELECT tool will not create a long running transaction. So, it is okay to set the ADTTIMEO to a value such as one minute.
ADTTOKEN	Data transfer token.
ADTVERS	Version number should be set to 2. The T09MCALL macro automatically sets the correct version number in the ADT parameter list.

Sample Program Usage

This partial program shows a method of non-blocking variable length receiving. A connection is made and the token is loaded from the CPT-ACM and used by all of the following Unicenter SOLVE:CPT service requests. The length of the expected data is unknown, and the length of the message buffer is moved into the ADTBUFFL field. ADTOPTNS is set to ADTNWAIT and ADTTMPRT for partial block receiving without waiting. Control returns to the program immediately.

The programmer determines if data was received by checking:

- The updated length in the ADTBUFFL field
- The return code is to determine RECEIVE service completion status

A return code of CPTWTIMO (1) in the ADTRTNCD field indicates the RECEIVE has timed out in the SELECT tool waiting for data.

The programmer should check the ADTRTNCD for CPTWBLCK (6), which indicates that the RECEIVE call will block. In the CPTWBLCK case, the program hands the connect token over to the SELECT tool by:

- Setting option AFMOPSEL
- Moving the transaction to kick off into field AFMNTRAN
- Calling the GIVE service as shown at label GIVESLCT

Important! *When transaction NXTR gets control again from the SELECT tool, the RECEIVE call must be issued with the exact same options set, otherwise the results are unpredictable.*

Note: Relevant parameters of the example are in **bold**.

```
*        Dsect's

         T09DACM MF=DSECT          Argument for Connection Management
         T09DADT MF=DSECT         Argument for RECV
         T09DAFM MF=DSECT          Argument for TAKE
*
*        Working storage
DFHEISTG DSECT
CPTPARMS DS     F                  CPT calling paramater
RECVARG  DS     XL(ADTLEN)        Argument for RECV
TAKEARG  DS     XL(AFMLEN)         Argument for GIVE/TAKE
CPTIOBUF DS     XL1024             CPT RECV/SEND BUFFER
ARGLENG  DS     H
*
label    DFHEIENT
*
         ...
*
     EXEC CICS RETRIEVE
          NOHANDLE
          SET  (ACMARG)
          LENGTH (R09)
     END-EXEC.
*
         CLC    ARGLENG,=AL2(ACMLEN)   HAVE WE GOT AN ACM BLOCK?
         BE     FIRSTTAK                 NO, PSEUDO CONVERSATIONAL
*                                            REENTRY WITH ONLY TOKEN
*
         L      R09,0(,R09)        GET TOKEN FROM COMMAREA
*                                  PASSED FROM SELECT TOOL
         B      COMMNTAK           JOIN COMMON TAKE LOGIC
*
*        TAKE THE CONNECTION FROM THE LISTENER
*
FIRSTTAK DS     0H
         L      R09,ACMTOKEN-CPTACM(,R09) GET TOKEN FROM COMMAREA ACM
*
COMMNTAK DS     0H
         USING AFM,TAKEARG
         ST     R09,AFMTOKEN       ASSOCIATE FACILITY MANAGEMENT
         ST     R09,ADTTOKEN       ASSOCIATE DATA TRANSFER
*
         LA     R01,TAKEARG        Point to control block
         ST     R01,CPTPARMS       Store address of control block
         T09MCALL TAKE,PARM=CPTPARMS
         LTR    R15,R15            Test Return Code
         BZ     TAKEOK             Good==>TAKEOK
*                                  Bad...
*        TAKE error from AFM
*
         L      R3,AFMRTNCD        Load GIVETAKE Return Code
         L      R4,AFMDGNCD        Load GIVETAKE Diagnostic Code
          ...
*
TAKEOK   DS     0H
RECV     DS     0H
         USING ADT,RECVARG
*
         ST     R9,ADTTOKEN        Save connection token
         LA     R04,CPTIOBUF        LOAD I/O BUFFER ADDRESS
         ST     R04,ADTBUFFA       SAVE RECEIVE BUFFER ADDRESS
         LA     R05,L'CPTIOBUF      LOAD I/O BUFFER LENGTH
         ST     R05,ADTBUFFL       SAVE RECEIVE BUFFER LENGTH
         LA     R03,60              LOAD Receive Timeout
         ST     R03,ADTTIMEO       SAVE RECEIVE Timeout
         MVI    ADTOPCD2,ADTNWAIT  No-Wait Use Select TOOL
         MVI    ADTOPCD1,ADTTMPRT  Timed parital Receive
```

```
*
         LA    R01,RECVARG         Point to control block
         ST    R01,CPTPARMS        Store address of control block
         T09MCALL RECV,PARM=CPTPARMS
         LTR   R15,R15             Test Return Code
         BZ    RECVOK               Good==>RECVOK Have data
*                                   What happened?
         CLC   ADTRTNCD,=AL4(CPTWBLCK)      Receive block?
         BE    GIVESLCT                     Yes==>Give to SELECT TOOL
         CLC   ADTRTNCD,=AL4(CPTERLSE)      No more data?
         BE    CONNCLOS                     Yes==>Connection CLOSEing
         CLC   ADTRTNCD,=AL4(CPTWTIMO)      Timeout occur?
         BE    RECV                         Yes==>RECV
*                                           Bad...
*        RECV error from ADT
*
         L     R3,ADTRTNCD          Load RECV Return Code
         L     R4,ADTDGNCD          Load RECV Diagnostic Code
          ...
*
RECVOK   DS    0H
          ...
*
GIVESLCT DS    0H
         USING AFM,TAKEARG
*
         ST    R9,AFMTOKEN         Save connection token
         MVC   AFMNTRAN,=C'NXTR'   NEXT TRANSACTION ID TO START
         MVI   AFMOPCD1,AFMOPSEL   GIVE TO SELECT TOOL
*
         LA    R01,TAKEARG         Point to control block
         ST    R01,CPTPARMS        Store address of control block
         T09MCALL GIVE,PARM=CPTPARMS
         LTR   R15,R15             Test Return Code
         BZ    GIVEOK               Good==>GIVEOK
*                                   Bad...
*        GIVE error from AFM
*
         L     R3,AFMRTNCD          Load GIVE Return Code
         L     R4,AFMDGNCD          Load GIVE Diagnostic Code
          ...
*
GIVEOK   DS    0H
*
         EXEC  CICS RETURN
```

Non-Blocking LL RECEIVE

A binary length contained in the first two characters of the received data stream determines the length of the expected data record. See the example in LL RECEIVE Option. When the RECEIVE call is issued with a no wait option, one of the following should occur under normal circumstances:

- The proper amount of data is available and returned to the caller.
- No data is returned with a will block (CPTWBLCK) reason code (6) set in the ADTRTNCD field of the ADT. When the wait condition is received, the caller gives the token over to the SELECT tool, which wakes the application up when more data is available.

Recommended ADT Parameters

The following table lists the recommended parameters for use by the Non-Blocking LL RECEIVE methodology. These parameters are set within the ADT control block; see Assembler DSECTs for sample information.

For a complete list of optional parameters, see Complete Parameter List.

Parameter	Description
ADTBUFFA	Set to address of user data area.
ADTBUFFL	Set to the maximum expected length of any record.
ADTOPCD1	Receiving method: set to ADTTYPLL.
ADTOPCD2	Receiving method: set to ADTNWAIT.
ADTTIMEO	RECEIVE timeout value, set to reasonable timeout for your network environment. Issuing RECEIVE calls with the SELECT tool does not create a long running transaction. So, it is okay to set the ADTTIMEO to a value such as one minute.
ADTTOKEN	Data transfer token.
ADTVERS	Version number should be set to 2. The T09MCALL macro automatically sets the correct version number in the ADT parameter list.

Sample Program Usage

This partial program shows a method of receiving records based on the first two bytes containing the length of the record.

A connection is made and the token is loaded from the CPT-ACM and used by all of the following Unicenter SOLVE:CPT service requests. The maximum length of the expected data record of 1024 is moved into the ADTBUFFL field. ADTOPTNS is set to ADTNWAIT and ADTTYPLL for imbedded length separator type receiving without waiting. Control is return to the program immediately.

The programmer determines if data was received by checking:

- The updated length in the ADTBUFFL field
- The return code is to determine RECEIVE service completion status

A return code of CPTWTIMO (1) in the ADTRTNCD field indicates the RECEIVE timed out in the SELECT tool waiting for data.

The programmer should check the ADTRTNCD for CPTWBLCK (6), which indicates that the RECEIVE call will block. In the CPTWBLCK case, the program hands the connect token over to the SELECT tool by:

- Setting option AFMOPSEL
- Moving the transaction to be kicked off into field AFMNTRAN
- Calling the GIVE service as shown at label GIVESLCT

Important! *When transaction* NXTR *gets control again from the SELECT tool, the RECEIVE call must be issued with the exact same options set, otherwise the results are unpredictable.*

Note: Relevant parameters of the example are in **bold**.

```
*        Dsect's

         T09DACM MF=DSECT          Argument for Connection Management
         T09DADT MF=DSECT         Argument for RECV
         T09DAFM MF=DSECT          Argument for TAKE
*
*        Working storage
DFHEISTG DSECT
CPTPARMS DS    F                   CPT calling paramater
RECVARG  DS    XL(ADTLEN)         Argument for RECV
TAKEARG  DS    XL(AFMLEN)          Argument for GIVE/TAKE
CPTIOBUF DS    XL1024              CPT RECV/SEND BUFFER
ARGLENG  DS    H
*
label    DFHEIENT
*
         ...
```

```
*
     EXEC CICS RETRIEVE
          NOHANDLE
          SET  (R09)
          LENGTH (ARGLENG)
     END-EXEC.
*
         CLC   ARGLENG,=AL2(ACMLEN)   HAVE WE GOT AN ACM BLOCK?
         BE    FIRSTTAK                 NO, PSEUDO CONVERSATIONAL
*                                           REENTRY WITH ONLY TOKEN
*
         L     R09,0(,R09)         GET TOKEN FROM COMMAREA
*                                  PASSED FROM SELECT TOOL
         B     COMMNTAK            JOIN COMMON TAKE LOGIC
*
*        TAKE THE CONNECTION FROM THE LISTENER
*
FIRSTTAK DS    0H
         L     R09,ACMTOKEN-CPTACM(,R09) GET TOKEN FROM COMMAREA ACM
*
COMMNTAK DS    0H
         USING AFM,TAKEARG
         ST    R09,AFMTOKEN        ASSOCIATE FACILITY MANAGEMENT
         ST    R09,ADTTOKEN        ASSOCIATE DATA TRANSFER
*
         LA    R01,TAKEARG         Point to control block
         ST    R01,CPTPARMS        Store address of control block
         T09MCALL TAKE,PARM=CPTPARMS
         LTR   R15,R15             Test Return Code
         BZ    TAKEOK              Good==>TAKEOK
*                                  Bad...
*        TAKE error from AFM
*
         L     R3,AFMRTNCD         Load GIVETAKE Return Code
         L     R4,AFMDGNCD         Load GIVETAKE Diagnostic Code
          ...
*
TAKEOK   DS    0H
RECV     DS    0H
         USING ADT,RECVARG
*
         ST    R9,ADTTOKEN         Save connection token
         LA    R04,CPTIOBUF         LOAD I/O BUFFER ADDRESS
         ST    R04,ADTBUFFA        SAVE RECEIVE BUFFER ADDRESS
         LA    R05,L'CPTIOBUF       LOAD I/O BUFFER LENGTH
         ST    R05,ADTBUFFL        SAVE RECEIVE BUFFER LENGTH
         LA    R03,60               LOAD Receive Timeout
         ST    R03,ADTTIMEO        SAVE RECEIVE Timeout
         MVI   ADTOPCD2,ADTNWAIT   No-Wait Use Select TOOL
         MVI   ADTOPCD1,ADTTYPLL   LL Receive
*
         LA    R01,RECVARG         Point to control block
         ST    R01,CPTPARMS        Store address of control block
         T09MCALL RECV,PARM=CPTPARMS
         LTR   R15,R15             Test Return Code
         BZ    RECVOK               Good==>RECVOK Have data
*                                   What happened?
         CLC   ADTRTNCD,=AL4(CPTWBLCK)     Receive block?
         BE    GIVESLCT                    Yes==>Give to SELECT TOOL
         CLC   ADTRTNCD,=AL4(CPTERLSE)     No more data?
         BE    CONNCLOS                    Yes==>Connection CLOSEing
         CLC   ADTRTNCD,=AL4(CPTWTIMO)     Timeout occur?
         BE    RECV                        Yes==>RECV
```

```
*                                           Bad...
*        RECV error from ADT
*
         L      R3,ADTRTNCD          Load RECV Return Code
         L      R4,ADTDGNCD          Load RECV Diagnostic Code
          ...
*
RECVOK   DS     0H
          ...
*
GIVESLCT DS     0H
         USING AFM,TAKEARG
*
         ST     R9,AFMTOKEN         Save connection token
         MVC    AFMNTRAN,=C'NXTR'   NEXT TRANSACTION ID TO START
         MVI    AFMOPCD1,AFMOPSEL   GIVE TO SELECT TOOL
*
         LA     R01,TAKEARG         Point to control block
         ST     R01,CPTPARMS        Store address of control block
         T09MCALL GIVE,PARM=CPTPARMS
         LTR    R15,R15             Test Return Code
         BZ     GIVEOK               Good==>GIVEOK
*                                    Bad...
*        GIVE error from AFM
*
         L      R3,AFMRTNCD          Load GIVE Return Code
         L      R4,AFMDGNCD          Load GIVE Diagnostic Code
          ...
*
GIVEOK   DS     0H
*
         EXEC  CICS RETURN
```

Non-Blocking Separator Character RECEIVE

The records are determined based on finding one or two separator characters. See the example in Separator Character RECEIVE Option section. When the RECEIVE call is issued with a no wait option, one of the following should occur under normal circumstances:

- The proper separator characters delimited record is available and returned to the caller.
- No data is returned with a will block (CPTWBLCK) reason code (6) set in the ADTRTNCD field of the ADT. When the wait condition is received, the caller gives the token over to the SELECT tool, which wakes the application up when more data is available.

Recommended ADT Parameters

The following table lists the recommended parameters for use by the Non-Blocking Separator Character RECEIVE methodology. These parameters are set within the ADT control block; see Assembler DSECTs for sample information.

For a complete list of optional parameters, see Complete Parameter List.

Parameter	Description
ADTBUFFA	Set to address of user data area.
ADTBUFFL	Set to the maximum expected length of any record.
ADTOPCD1	Receiving method: set to ADTTYPSP.
ADTOPCD2	Receiving method: set to ADTNWAIT.
ADTSEP#	Set to 1 or 2 for the number of separator characters.
ADTSEP1	First or only separator character.
ADTSEP2	Second separator character in a sequence of two.
ADTTIMEO	RECEIVE timeout value, set to reasonable timeout for your network environment. Issuing RECEIVE calls with the SELECT tool does not create a long running transaction. So, it is okay to set the ADTTIMEO to a value such as one minute.
ADTTOKEN	Data transfer token.
ADTVERS	Version number should be set to 2. The T09MCALL macro automatically sets the correct version number in the ADT parameter list.

Sample Program Usage

This partial program shows a method of non-blocking separator character delineated receive. The records are determined based on finding one or two separator characters.

A connection is made and the token is loaded from the ACM and used by all of the following Unicenter SOLVE:CPT service requests. The maximum length of the expected data of 1024 is moved into the ADTBUFFL field. ADTOPTNS is set to ADTNWAIT and ADTTYPSP for separator character delimiters with no waiting. Control returns to the program immediately.

The programmer will:

- Determine if data was received by checking the updated length in the ADTBUFFL field
- Check that the return code is to determine RECEIVE service completion status

A return code of CPTWTIMO (1) in the ADTRTNCD field indicates the RECEIVE timed out in the SELECT tool waiting for data.

The programmer should check the ADTRTNCD for CPTWBLCK (6), which indicates that the RECEIVE call will block. In the CPTWBLCK case, the program hands the connect token over to the SELECT tool by:

- Setting option AFMOPSEL
- Moving the transaction to be kicked off into field AFMNTRAN
- Calling the GIVE service as shown at label GIVESLCT

When transaction *my-tran* gets control again from the SELECT tool, the RECEIVE call must be issued with the exact same options set, otherwise the results are unpredictable.

Note: Relevant parameters of the example are in **bold**.

```
*        Dsect's

         T09DACM MF=DSECT          Argument for Connection Management
         T09DADT MF=DSECT         Argument for RECV
         T09DAFM MF=DSECT          Argument for TAKE
*
*        Working storage
DFHEISTG DSECT
CPTPARMS DS    F                   CPT calling paramater
RECVARG  DS    XL(ADTLEN)         Argument for RECV
TAKEARG  DS    XL(AFMLEN)          Argument for GIVE/TAKE
CPTIOBUF DS    XL1024              CPT RECV/SEND BUFFER
ARGLENG  DS    H
*
label    DFHEIENT
*
         ...
*
     EXEC CICS RETRIEVE
          NOHANDLE
          SET  (ACMARG)
          LENGTH (ARGLENG)
     END-EXEC.
*
         CLC   ARGLENG,=AL2(ACMLEN)   HAVE WE GOT AN ACM BLOCK?
         BE    FIRSTTAK                  NO, PSEUDO CONVERSATIONAL
*                                            REENTRY WITH ONLY TOKEN
*
         L     R09,0(,R09)         GET TOKEN FROM COMMAREA
*                                  PASSED FROM SELECT TOOL
         B     COMMNTAK            JOIN COMMON TAKE LOGIC
*
*        TAKE THE CONNECTION FROM THE LISTENER
*
FIRSTTAK DS    0H
         L     R09,ACMTOKEN-CPTACM(,R09) GET TOKEN FROM COMMAREA ACM
*
COMMNTAK DS    0H
         USING AFM,TAKEARG
         ST    R09,AFMTOKEN        ASSOCIATE FACILITY MANAGEMENT
         ST    R09,ADTTOKEN        ASSOCIATE DATA TRANSFER
*
         LA    R01,TAKEARG         Point to control block
         ST    R01,CPTPARMS        Store address of control block
         T09MCALL TAKE,PARM=CPTPARMS
         LTR   R15,R15             Test Return Code
         BZ    TAKEOK              Good==>TAKEOK
*                                  Bad...
*        TAKE error from AFM
*
         L     R3,AFMRTNCD         Load GIVETAKE Return Code
         L     R4,AFMDGNCD         Load GIVETAKE Diagnostic Code
          ...
*
TAKEOK   DS    0H
RECV     DS    0H
         USING ADT,RECVARG
*
         ST    R9,ADTTOKEN         Save connection token
         LA    R04,CPTIOBUF         LOAD I/O BUFFER ADDRESS
         ST    R04,ADTBUFFA        SAVE RECEIVE BUFFER ADDRESS
         LA    R05,L'CPTIOBUF       LOAD I/O BUFFER LENGTH
         ST    R05,ADTBUFFL        SAVE RECEIVE BUFFER LENGTH
```

```
         LA    R03,60              LOAD Receive Timeout
         ST    R03,ADTTIMEO        SAVE RECEIVE Timeout
         MVI   ADTOPCD2,ADTNWAIT   No-Wait Use Select TOOL
         MVI   ADTOPCD1,ADTTYPSP   Separator Character Receive
         MVI   ADTSEP#,2           2 Separator Characters
         MVI   ADTSEP1,X’0D’       CR is 1st Separator Character
         MVI   ADTSEP1,X’0A’       LF is 2nd Separator Character
*
         LA    R01,RECVARG         Point to control block
         ST    R01,CPTPARMS        Store address of control block
         T09MCALL RECV,PARM=CPTPARMS
         LTR   R15,R15             Test Return Code
         BZ    RECVOK              Good==>RECVOK Have data
*                                  What happened?
         CLC   ADTRTNCD,=AL4(CPTWBLCK)     Receive block?
         BE    GIVESLCT                    Yes==>Give to SELECT TOOL
         CLC   ADTRTNCD,=AL4(CPTERLSE)     No more data?
         BE    CONNCLOS                    Yes==>Connection CLOSEing
         CLC   ADTRTNCD,=AL4(CPTWTIMO)     Timeout occur?
         BE    RECV                        Yes==>RECV
*                                          Bad...
*        RECV error from ADT
*
         L     R3,ADTRTNCD         Load RECV Return Code
         L     R4,ADTDGNCD         Load RECV Diagnostic Code
          ...
*
RECVOK   DS    0H
          ...
*
GIVESLCT DS    0H
         USING AFM,TAKEARG
*
         ST    R9,AFMTOKEN         Save connection token
         MVC   AFMNTRAN,=C'NXTR'   NEXT TRANSACTION ID TO START
         MVI   AFMOPCD1,AFMOPSEL   GIVE TO SELECT TOOL
*
         LA    R01,TAKEARG         Point to control block
         ST    R01,CPTPARMS        Store address of control block
         T09MCALL GIVE,PARM=CPTPARMS
         LTR   R15,R15             Test Return Code
         BZ    GIVEOK              Good==>GIVEOK
*                                  Bad...
*        GIVE error from AFM
*
         L     R3,AFMRTNCD         Load GIVE Return Code
         L     R4,AFMDGNCD         Load GIVE Diagnostic Code
          ...
*
GIVEOK   DS    0H
*
         EXEC  CICS RETURN
```

Blocking Fixed Length RECEIVE

The length of the expected data is known and the receive waits until all data is received or the ADTTIMEO timeout expires.

Recommended ADT Parameters

The following table lists the recommended parameters for use by the Blocking Fixed Length RECEIVE methodology. These parameters are set within the ADT control block; see Assembler DSECTs for sample information.

For a complete list of optional parameters, see Complete Parameter List.

Parameter	Description
ADTBUFFA	Set to address of user data area.
ADTBUFFL	Set to expected fixed length record.
ADTOPCD1	Receiving method: set to ADTTMRCV.
ADTTIMEO	RECEIVE timeout value, set to reasonable timeout for your network environment.
ADTTOKEN	Data transfer token.
ADTVERS	Version number should be set to 2. The T09MCALL macro automatically sets the correct version number in the ADT parameter list.

Sample Program Usage

This partial program shows a method of receiving fixed length records while blocking the connection. A connection is made and the token is loaded from the ACM and used by all of the following Unicenter SOLVE:CPT service requests. The length of the expected data is known, and 345 is moved into the ADTBUFFL field. ADTOPTNS is set to ADTTMRCV for full block receiving.

The receive waits until all data is received or the ADTTIMEO timeout expires. The ADTRTNCD return code is checked to determine RECEIVE service completion status.

Note: Relevant parameters of the example are in **bold.**

```
*        Dsect's
         T09DACM MF=DSECT         Argument for Connection Management
         T09DADT MF=DSECT         Argument for RECV
*
*        Working storage
DFHEISTG DSECT
CPTPARMS DS     F                 CPT calling paramater
RECVARG  DS     XL(ADTLEN)        Argument for RECV
CPTIOBUF DS     XL1024            CPT RECV/SEND BUFFER
ARGLENG  DS     H
*
label    DFHEIENT
*
         ...
*
          L     R9,ACMTOKEN       Load ACM Token
*
RECV     DS     0H
         USING ADT,RECVARG
*
         ST     R9,ADTTOKEN       Save connection token
         LA     R04,CPTIOBUF       LOAD I/O BUFFER ADDRESS
         ST     R04,ADTBUFFA      SAVE RECEIVE BUFFER ADDRESS
         LA     R05,345            LOAD known data length
         ST     R05,ADTBUFFL      SAVE RECEIVE BUFFER LENGTH
         LA     R03,5              LOAD Receive Timeout
         ST     R03,ADTTIMEO      SAVE RECEIVE Timeout
         MVI    ADTOPCD1,ADTTMRCV Timed Receive
*
         LA     R01,RECVARG       Point to control block
         ST     R01,CPTPARMS      Store address of control block
         T09MCALL RECV,PARM=CPTPARMS
         LTR    R15,R15           Test Return Code
         BZ     RECVOK             Good==>RECVOK Have data
*                                  What happened?
         CLC    ADTRTNCD,=AL4(CPTERLSE)     No more data?
         BE     CONNCLOS                    Yes==>Connection CLOSEing
         CLC    ADTRTNCD,=AL4(CPTWTIMO)     Timeout occur?
         BE     RECV                        Yes==>RECV
*                                           Bad...
*        RECV error from ADT
*
         L      R3,ADTRTNCD        Load RECV Return Code
         L      R4,ADTDGNCD        Load RECV Diagnostic Code
          ...
*
RECVOK   DS     0H
*
```

Blocking LL RECEIVE

A binary length, LL, contained in the first two characters of the received data stream determines the length of the expected data record. See example in LL RECEIVE Option The LL length is used on a RECEIVE call which waits until all data has been received or the ADTTIMEO timeout expires.

Recommended ADT Parameters

The following table lists the recommended parameters for use by the Blocking LL RECEIVE methodology. These parameters are set within the ADT control block; see Assembler DSECTs for sample information.

For a complete list of optional parameters see Complete Parameter List.

Parameter	Description
ADTBUFFA	Set to address of user data area.
ADTBUFFL	Set to maximum expected length of any record.
ADTOPCD1	Receiving method: set to ADTTYPLL.
ADTTIMEO	RECEIVE timeout value, set to reasonable timeout for your network environment.
ADTTOKEN	Data transfer token.
ADTVERS	Version number should be set to 2. The T09MCALL macro automatically sets the correct version number in the ADT parameter list.

Sample Program Usage

This partial program shows a method of receiving records based on the first two bytes containing the length of the record. A connection is made and the token is loaded from the ACM and used by all of the following Unicenter SOLVE:CPT service requests. The maximum length of the expected data record of 1024 is moved into the ADTBUFFL field. ADTOPTNS is set to ADTTYPLL for imbedded length separator type receiving. The binary length contained in the first two characters of the received data stream determines the length of the expected data.

The receive waits until all data is received or a ADTTIMEO timeout expires. The return code is checked to determine RECEIVE service completion status.

Note: Relevant parameters of the example are in **bold.**

```
*        Dsect's
         T09DACM MF=DSECT          Argument for Connection Management
         T09DADT MF=DSECT         Argument for RECV
*
*        Working storage
DFHEISTG DSECT
CPTPARMS DS     F                  CPT calling paramater
RECVARG  DS     XL(ADTLEN)        Argument for RECV
CPTIOBUF DS     XL1024             CPT RECV/SEND BUFFER
ARGLENG  DS     H
*
label    DFHEIENT
*
         ...
*
          L     R9,ACMTOKEN        Load ACM Token
*
RECV     DS     0H
         USING ADT,RECVARG
*
         ST     R9,ADTTOKEN        Save connection token
         LA     R04,CPTIOBUF        LOAD I/O BUFFER ADDRESS
         ST     R04,ADTBUFFA       SAVE RECEIVE BUFFER ADDRESS
         LA     R05,L'CPTIOBUF      LOAD buffer size
         ST     R05,ADTBUFFL       SAVE RECEIVE BUFFER LENGTH
         LA     R03,5               LOAD Receive Timeout
         ST     R03,ADTTIMEO       SAVE RECEIVE Timeout
         MVI    ADTOPCD1,ADTTYPLL  LL Receive
*
         LA     R01,RECVARG        Point to control block
         ST     R01,CPTPARMS       Store address of control block
         T09MCALL RECV,PARM=CPTPARMS
         LTR    R15,R15            Test Return Code
         BZ     RECVOK              Good==>RECVOK Have data
*                                   What happened?
         CLC    ADTRTNCD,=AL4(CPTERLSE)    No more data?
         BE     CONNCLOS                   Yes==>Connection CLOSEing
         CLC    ADTRTNCD,=AL4(CPTWTIMO)    Timeout occur?
         BE     RECV                       Yes==>RECV
*                                          Bad...
*        RECV error from ADT
*
         L      R3,ADTRTNCD         Load RECV Return Code
         L      R4,ADTDGNCD         Load RECV Diagnostic Code
          ...
*
RECVOK   DS     0H
*
```

Blocking Separator Character RECEIVE

The records are determined based on finding one or two separator characters. See the example in Separator Character RECEIVE Option section. The data is received until the separator characters are found, and then the record is given to the RECEIVE caller or the ADTTIMEO timeout expires.

Recommended ADT Parameters

The following table lists the recommended parameters for use by the Blocking Separator Character RECEIVE methodology. These parameters are set within the ADT control block; see Assembler DSECTs for sample information.

For a complete list of optional parameters, see Complete Parameter List.

Parameter	Description
ADTBUFFA	Set to address of user data area.
ADTBUFFL	Set to the maximum expected length of any record.
ADTOPCD1	Receiving method: set to ADTTYPSP.
ADTSEP1	First or only separator character.
ADTSEP2	Second separator character in a sequence of two.
ADTSEP#	Set to 1 or 2 for the number of separator characters.
ADTTIMEO	RECEIVE timeout value, set to reasonable timeout for your network environment.
ADTTOKEN	Data transfer token.
ADTVERS	Version number should be set to 2. The T09MCALL macro automatically sets the correct version number in the ADT parameter list.

Sample Program Usage

This partial program shows a method of implementing the Blocked Separator Character RECEIVE. The records are determined based on finding one or two separator characters.

A connection is made and the token is loaded from the ACM and used by all of the following Unicenter SOLVE:CPT service requests. The maximum length of the expected data of 1024 is moved into the ADTBUFFL field. ADTOPTNS is set to ADTTYPSP for separator character delimiters.

The Blocked Separator Character RECEIVE waits up to the ADTTIMEO expiration for all the data to be received. The ADTRTNCD return code is checked to determine RECEIVE service completion status.

Note: Relevant parameters of the example are in **bold.**

```
*        Dsect's

         T09DACM MF=DSECT          Argument for Connection Management
         T09DADT MF=DSECT         Argument for RECV
*
*        Working storage
DFHEISTG DSECT
CPTPARMS DS    F                   CPT calling paramater
RECVARG  DS    XL(ADTLEN)         Argument for RECV
CPTIOBUF DS    XL1024              CPT RECV/SEND BUFFER

*
label    DFHEIENT
*
         ...
*
          L    R9,ACMTOKEN         Load ACM Token
*
RECV     DS    0H
         USING ADT,RECVARG
*
         ST    R9,ADTTOKEN         Save connection token
         LA    R04,CPTIOBUF         LOAD I/O BUFFER ADDRESS
         ST    R04,ADTBUFFA        SAVE RECEIVE BUFFER ADDRESS
         LA    R05,L'CPTIOBUF       LOAD buffer size
         ST    R05,ADTBUFFL        SAVE RECEIVE BUFFER LENGTH
         LA    R03,5                LOAD Receive Timeout
         ST    R03,ADTTIMEO        SAVE RECEIVE Timeout
         MVI   ADTOPCD1,ADTTYPSP   Separator Character Receive
         MVI   ADTSEP#,2           2 Separator Characters
         MVI   ADTSEP1,X'0D'       CR is 1st Separator Character
         MVI   ADTSEP1,X'0A'       LF is 2nd Separator Character
*
         LA    R01,RECVARG         Point to control block
         ST    R01,CPTPARMS        Store address of control block
         T09MCALL RECV,PARM=CPTPARMS
         LTR   R15,R15             Test Return Code
         BZ    RECVOK               Good==>RECVOK Have data
*                                   What happened?
         CLC   ADTRTNCD,=AL4(CPTERLSE)      No more data?
         BE    CONNCLOS                     Yes==>Connection CLOSEing
         CLC   ADTRTNCD,=AL4(CPTWTIMO)      Timeout occur?
         BE    RECV                         Yes==>RECV
```

```
*                                          Bad...
*        RECV error from ADT
*
         L     R3,ADTRTNCD          Load RECV Return Code
         L     R4,ADTDGNCD          Load RECV Diagnostic Code
          ...
*
RECVOK   DS    0H
```

Non-Blocking RECEIVE

In this method, the application continuously issues no wait RECEIVE calls holding the data and looping back to receive more data until the data is exhausted or the application determines that it has what it needs.

Recommended ADT Parameters

The following table lists the recommended parameters for use by the Non-Blocking Receive Loop methodology. These parameters are set within the ADT control block; see Assembler DSECTs for sample information.

For a complete list of optional parameters see Complete Parameter List.

Parameter	Description
ADTBUFFA	Set to address of user data area.
ADTBUFFL	Set to expected maximum length of record.
ADTOPCD1	Receiving method: set to ADTNBLKR.
ADTTOKEN	Data transfer token.
ADTVERS	Version number should be set to 2. The T09MCALL macro automatically sets the correct version number in the ADT parameter list.

Sample Program Usage

This partial program shows an older non-blocking receive method. This is a polling method to check to see if any data is at the endpoint.

A connection is made and the token is loaded from the CPT-ACM and used by all of the following Unicenter SOLVE:CPT service requests. The maximum length of the expected data of 1024 is moved into the ADTBUFFL field. ADTOPTNS is set to ADTNBLKR for standard sockets in a non-blocking receive mode.

The RECEIVE service always returns control back to the caller. If no data is available, then return code CPTWBLCK (6) is returned in the ADTRTNCD field of the ADT. The return code is checked to determine RECEIVE service completion status. At this point, it is the programmer's responsibility to check the ADTBUFFL to determine if any data was received and how to process it

Note: Relevant parameters of the example are in **bold.**

```
*        Dsect's

         T09DACM MF=DSECT          Argument for Connection Management
         T09DADT MF=DSECT         Argument for RECV
*
*        Working storage
DFHEISTG DSECT
CPTPARMS DS    F                   CPT calling paramater
RECVARG  DS    XL(ADTLEN)         Argument for RECV
CPTIOBUF DS    XL1024              CPT RECV/SEND BUFFER
*
label    DFHEIENT
*
         ...
*
          L    R9,ACMTOKEN         Load ACM Token
*
RECV     DS    0H
         USING ADT,RECVARG
*
         ST    R9,ADTTOKEN         Save connection token
         LA    R04,CPTIOBUF         LOAD I/O BUFFER ADDRESS
         ST    R04,ADTBUFFA        SAVE RECEIVE BUFFER ADDRESS
         LA    R05,L'CPTIOBUF       LOAD buffer size
         ST    R05,ADTBUFFL        SAVE RECEIVE BUFFER LENGTH
         MVI   ADTOPCD1,ADTNBLKR   Non-Blocking Receive
*
         LA    R01,RECVARG         Point to control block
         ST    R01,CPTPARMS        Store address of control block
         T09MCALL RECV,PARM=CPTPARMS
         LTR   R15,R15             Test Return Code
         BZ    RECVOK               Good==>RECVOK Have data
*                                   What happened?
         CLC   ADTRTNCD,=AL4(CPTERLSE)      No more data?
         BE    CONNCLOS                     Yes==>Connection CLOSEing
         CLC   ADTRTNCD,=AL4(CPTWBLCK)      Receive block?
         BE    NODATA                       Yes==>NODATA
*                                           Bad...
*        RECV error from ADT
*
         L     R3,ADTRTNCD          Load RECV Return Code
         L     R4,ADTDGNCD          Load RECV Diagnostic Code
          ...
*
RECVOK   DS    0H
```

Blocking RECEIVE

RECEIVE is called for data. The RECEIVE call can become a long running task waiting for network data to arrive.

What makes this a blocking receive call is that the call must receive some data (or a failure at the transport provider) or control is never return to the caller. That is why in the recommendation below we suggest a timeout.

Recommended ADT Parameters

The following table lists the recommended parameters for use by the Blocking Receive Loop methodology. These parameters are set within the ADT control block; see Assembler DSECTs for sample information.

For a complete list of optional parameters, see Complete Parameter List.

Parameter	Description
ADTBUFFA	Set to address of user data area.
ADTBUFFL	Set to maximum expected length of record.
ADTOPCD1	Receiving method: set to ADTTMPRT.
ADTTIMEO	RECEIVE timeout value, set to reasonable timeout for your network environment.
ADTTOKEN	Data transfer token.
ADTVERS	Version number should be set to 2. The T09MCALL macro automatically sets the correct version number in the ADT parameter list.

Sample Program Usage

This partial program shows a blocking receive. A connection is made and the token is loaded from the ACM and used by all of the following Unicenter SOLVE:CPT service requests. The maximum length of the expected data of 1024 is moved into the ADTBUFFL field. ADTOPTNS is set to ADTNTMPRT.

The RECEIVE service returns control back to the caller

- Any time data is available
- The transport provider detects an error
- An ADTTIMEO timeout expires

The return code is checked to determine RECEIVE service completion status. At this point, it is the programmer's responsibility to check the ADTBUFFL to determine if any data was received and how to process it.

Note: Relevant parameters of the example are in **bold.**

```
*        Dsect's

         T09DACM MF=DSECT          Argument for Connection Management
         T09DADT MF=DSECT         Argument for RECV
*
*        Working storage
DFHEISTG DSECT
CPTPARMS DS    F                   CPT calling paramater
RECVARG  DS    XL(ADTLEN)         Argument for RECV
CPTIOBUF DS    XL1024              CPT RECV/SEND BUFFER
ARGLENG  DS    H
*
label    DFHEIENT
*
         ...
*
          L    R9,ACMTOKEN         Load ACM Token
*
RECV     DS    0H
         USING ADT,RECVARG
*
         ST    R9,ADTTOKEN         Save connection token
         LA    R04,CPTIOBUF         LOAD I/O BUFFER ADDRESS
         ST    R04,ADTBUFFA        SAVE RECEIVE BUFFER ADDRESS
         LA    R05,L'CPTIOBUF       LOAD buffer size
         ST    R05,ADTBUFFL        SAVE RECEIVE BUFFER LENGTH
         LA    R03,5                LOAD Receive Timeout
         ST    R03,ADTTIMEO        SAVE RECEIVE Timeout
         MVI   ADTOPCD1,ADTTMPRT   Iimed partial*
         LA    R01,RECVARG         Point to control block
         ST    R01,CPTPARMS        Store address of control block
         T09MCALL RECV,PARM=CPTPARMS
         LTR   R15,R15             Test Return Code
         BZ    RECVOK               Good==>RECVOK Have data
*                                   What happened?
         CLC   ADTRTNCD,=AL4(CPTERLSE)     No more data?
         BE    CONNCLOS                    Yes==>Connection CLOSEing
         CLC   ADTRTNCD,=AL4(CPTWTIMO)     Timeout occur?
         BE    RECV                        Yes==>RECV
```

```
*                                          Bad...
*        RECV error from ADT
*
         L     R3,ADTRTNCD          Load RECV Return Code
         L     R4,ADTDGNCD          Load RECV Diagnostic Code
          ...
*
RECVOK   DS    0H
```

Parameter Values Returned in the ADT

After the RECEIVE call returns control to your application program, the following fields are propagated with valid information. These updated values are passed back to the application in the ADT control block.

Parameter	Description
ADTBUFFA	Data buffer filled with data from the network.
ADTBUFFL	Length of the data received.
ADTDGNCD	Diagnostic code.
ADTRTNCD	Return code.

Assembler DSECTs

Sample Assembler DSECTs are provided in the distributed software and are available to you in ***cpthlq***.T09MAC. Variable field names contained in the distributed samples and the examples in this guide refer to these DSECTs.

T09DADT — Assembler DSECT name for the ADT. For detailed information and a sample copy of the Assembler DSECT, see the ADT: Argument for Data Transfer Used by RECIEVE, SEND, RECVFROM, and SENDTO Services Service section in appendix “Control Block Layouts.”

All Assembler constants that apply to ADT calls are imbedded in the ADT DSECT sample.

Sample Programs

Sample Assembler source code is available in the distributed software in the ***cpthlq***.T09SAMP library. You should be able to find a sample that matches your programming requirement. For complete details on the function a sample program provides, see the program descriptions in the "Unicenter SOLVE:CPT API Services" chapter and the descriptions of the sample members listed below.

Name	Description
T09PACL1	Client Application sends typed in data to the server waiting for the information to be echoed back from the server.
T09PACL2	Client Application to send an internal message using either the FULL, SEP or LL to be echoed back by the server.
T09PASV1	TCP Server 1 program is a single-threaded server using a Listen API call.
T09PASV2	TCP Server 2 program is a multithreaded server using the Listen tool.
T09PASV5	TCP Server 5 program is a multithreaded server using both the Listen and SELECT tools.

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, unless overridden with selected ADT options. 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, unless overridden with selected ADT options.

In the case of specifying a LL or separator type RECEIVE, completion will not occur until all the data of length LL (for LL receive), the separators have been found (for separator receive), or a timeout occurs. This may cause the transaction to wait within the RECEIVE service unless the ADTNWAIT option is used.

The presence of exceptions or error conditions does 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 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 and the diagnostic code indicate the nature of the failure. The diagnostic code generally contains a specific code that is generated by the transport provider.

Return Codes

The RECEIVE service returns a code in registers R15 and R0 indicating the results of the execution. These values are in the ADTRTNCD (R15) and ADTDGNCD (R0).

Sample Assembler DSECT T09DRTCD is provided in data set ***cpthlq***.T09MAC. It details the variable field names contained in the distributed samples and the examples in this guide. See the appendix "Return Codes" for a sample copy of the T09DRTCD DSECT. A description of the problem causing the associated return code is contained in this DSECT.

The following table lists the return codes that can apply to the RECEIVE call.

Decimal	Hex	Diagnostic Code	Variable	Description
0	0	No	CPTIRCOK	Request completed successfully.
1	1	No	CPTWTIMO	Timed receive call timed out.
6	6	Yes	CPTWBLCK	Non-blocking call to the REVEIVE service detected a wait condition.
17	11	No	CPTEVRSN	Control block version number not supported.
20	14	No	CPTETOKN	Specified data transfer token is invalid.
21	15	No	CPTEBUFF	Buffer address or length invalid.
27	1B	No	CPTETIME	Receive timeout value not specified.
31	1F	No	CPTEFRMT	Other Socket Call Parameter List format or specification error.
34	22	No	CPTENAPI	API not fully available; retry.
40	28	Yes	CPTETERM	TCPIP is terminating.
7	2F	Yes	CPTEENVR	Other transport layer environmental condition.

Decimal	Hex	Diagnostic Code	Variable	Description
65	41	Yes	CPTERLSE	Orderly release of remote connection request.
68	44	Yes	CPTEDISC	Remote connection not available or aborted.
79	4F	Yes	CPTEINTG	Other transport layer connection/data integrity error.
143	8F	Yes	CPTEPROC	Procedural error.
254	FE	Is abend code	CPTABEND	Abnormal termination. The diagnostic code is the abnormal termination code, which is normally a CICS abend code, but can also be in the "Abend Codes" chapter of the *Message Guide*.
255	FF	No	CPTEOTHR	Other error.

Usage Notes

The RECEIVE service receives normal data inputs through a Unicenter SOLVE:CPT connection. The data may be part of a byte stream being received over a connection (TCP).

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.

A user application 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 ADTOPTNS field specifies RECEIVE processing control options. These options provide the application more flexibility and make it easier to deal with stream data than through the default blocking RECEIVE option. ADTNOWAIT allows for polling of a connection for data or for use with the SELECT service. ADTTMRCV and ADTTMPRT provide logical request capabilities to the RECEIVE service. The options ADTTYPSP and ADTTYPLL allow for built-in record delimiters

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 ADTVERS version number indicates the Unicenter SOLVE: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 ADTFUNC function code indicates the Unicenter SOLVE: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, specifies 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 ADTBUFFL field indicates the data buffer length. This is a full word unsigned 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.

Complete Parameter List

ADTBUFFA	Required. 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: None.
ADTBUFFL	Required. 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: None.
ADTDGNCD	Diagnostic code. Indicates the diagnostic code set by the service request. This value generally indicates a transport provider return code. 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
ADTLADDR	Used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
ADTLNAME	Used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
ADTLPORT	Used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
ADTMRECV	Used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.

ADTMSEND — Used only by the UDP calls RCVFROM and SENDTO.

For TCP connections, this parameter is set in the equivalent ACM field.

ADTMSOCK — Has no meaning for TCP connections, since the maximum number of sockets is set at connection establishment time through the ACMMSOCK field.

ADTNSLCT — Number of entries in the selected vector. Not used by the RECEIVE service.

ADTOPCD1 — Specifies byte one(1) of data transfer options.

These are the ADT options that apply to TCP data transfer requests:

It is an error to just combine any of these RECEIVE service options:

ADTNBLKR	ADTTYPLL	ADTTYPSP
ADTTMRCV	ADTTMPRT	

Important! *An invalid combination will result in CPTEOPTN being returned in ADTRTNCD.*

ADTBLCKS — This option was disabled with the CPT 6.1 API conversion. This option is ignored.

ADTFDNR — Do DNR name resolution (not used by the SEND service).

ADTNBLKR — Do not block on a call to the RECEIVE service. If no data is currently available on the connection, CPTWBLCK is returned in ADTRTNCD.

This token can subsequently be passed to the SELECT tool. See SELECT tool.

ADTTMPRT — Timed partial record RECEIVE.

These fields (along with other required ADT fields) are used to request a timed partial record RECEIVE:

ADTBUFFL set to maximum length expected.

ADTTMPRT

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, is returned to the caller.

ADTTMRCV	Timed full record RECEIVE. These fields (along with other required ADT fields) are used to request a timed full record RECEIVE: ADTBUFFL set to the length expected ADTTMRCV ADTTIMEO > zero If the time limit expires before receiving any or all of the data specified by ADTBUFFL, CPTWTIMO is returned in ADTRTNCD along with any data that was received.
ADTTYPLL	LL type RECEIVE, see example in section: LL RECEIVE Option These fields (along with other required ADT fields) are used to request a SEP type RECEIVE call: 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 are returned in ADTRTNCD along with any data that was received.
ADTTYPSP	SEP type RECEIVE, see example in section: Separator Character RECEIVE Option These fields (along with other required ADT fields) are used to request a SEP type RECEIVE call: 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 is returned to the caller.

Default: None.

ADTOPCD2	Specifies byte two(2) of data transfer options.

These are the ADT options that apply to TCP data transfer requests:

Important! *An invalid combination will result in CPTEOPTN being returned in ADTRTNCD.*

ADTFVLST	Currently for internal use only.
ADTNOQUE	Do not QUEUE API RECEIVES.
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.
ADTNWAIT	Do not wait on a call to the RECEIVE service. If no data is currently available on the connection, CPTWBLCK is returned in ADTRTNCD. The token can subsequently be passed to the SELECT tool. See The SELECT tool in the chapter "Unicenter SOLVE:CPT Tools." This differs from ADTNBLKR since ADTNWAIT can be used for all types of receives (timed, separator, LL) whereas ADTNBLKR is a normal non-blocking stream receive.
ADTRT100	The ADTTIMEO value is expressed in 1/100 of a second. When flag ADTRT100 is **not set** in the ADTOPCD2 field then field ADTTIMEO specifies the amount of time in seconds to wait for data to be received. When flag ADTRT100 is **set** in the ADTOPCD2 field then field ADTTIMEO specifies the amount of time in 1/100 seconds to wait for data to be received.

Default: None.

ADTOPTNS	Specifies data transfer options. See ADTOPCD1 and ADTOPCD2 above.
ADTQRECV	Used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
ADTQSEND	Used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
ADTRADDR	Used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
ADTRNAME	Used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
ADTRPORT	Used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
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: None.
ADTSEP#	Number of separator characters for option ADTTYPSP. When option ADTTYPSP has been set then the ADTSEP# must be either one or two or CPTESEP# will be returned in the ADTRTNCD field. 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.
ADTSLCTD	Number of tokens selected. Not used by the RECEIVE service.
ADTSRVCE	Used only by the UDP calls: RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.

ADTSTAT Specifies statistics logging options for the application program.

ADTSCONN	Specifies that messages be generated on the initial connection of a session.
ADTSTERM	Specifies that messages be generated on terminating an established connection.

Default: None, no statistics logging.

ADTTIMEO RECEIVE timeout value.

Must be specified with these options:

- ADTTMPRT
- ADTTMRCV
- ADTTYPLL
- ADTTYPSP

Specifying any of the above options on a RECEIVE call with an ADTTIMEO value of zero (0) results in CPTETIME being returned in ADTRTNCD.

When flag ADTRT100 is **not set** in the ADTOPCD2 field then field ADTTIMEO specifies the amount of time in seconds to wait for data to be received.

When flag ADTRT100 is **set** in the ADTOPCD2 field then field ADTTIMEO specifies the amount of time in 1/100 seconds to wait for data to be received.

A CPTWTIMO error occurs when the data is not received by the ADTTIMEO timeout.

Default: None.

ADTTOKEN Required. Data transfer token. ADTTOKEN 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, using 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: None.

ADTTRACE	**Note:** The tracing functionality has moved in version 6 of Unicenter SOLVE:CPT. A greatly enhanced tracing capability is now available using the TCPEEP tracing command. See the *Administrator Guide* for more detail.
	These tracing fields remain only for downward compatibility purposes and are ignored.

ACMTNTRY	ACMTTERM	ACMTTPL
ACMTARGS	ACMTPASS	ACMTRLSE
ACMTRECV	ACMTCLSE	ACMTSTOR
ACMTSEND	ACMTTERR	ACMTCLTD

ADTUCNTX	Used only by the UDP calls RCVFROM and SENDTO.
	For TCP connections, this parameter is set in the equivalent ACM field.
ADTVECTR	Address of the selected vector. Not used by the RECEIVE service.
ADTVERS	Required version number. Indicates the Unicenter SOLVE:CPT version number of the argument used by the calling program.
	Must be set to a binary two for this release of Unicenter SOLVE:CPT.
	Default: None.

Chapter

10 SEND Service

The SEND service sends data to a peer transport user connected to an endpoint. It also 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 (Argument for Data Transfer) 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 chapter discusses the following topics:

- Call Syntax—Shows sample syntax for the SEND service call
- Recommended ADT Parameters—Lists the parameters normally used and recommended for the SEND service call
- Usage Examples—Provides a sample program shell for using the SEND service call
- Parameter Values Returned in the ADT—List fields that are updated in the ADT control block upon return from the SEND service call
- Assembler DSECTs— Lists information about the distributed sample Assembler DSECTs that are used by the SEND service call
- Sample Programs—Sample Assembler programs that use the SEND service.
- Completion Information—Describes the expected results at completion of the SEND service call
- Return Codes—Lists the return codes that can apply to the SEND service call
- Usage Notes—Contains miscellaneous notes about usage of the SEND service call
- Complete Parameter List—Provides a complete list of the parameters and their options for the SEND service call

Call Syntax

Computer Associates recommends that a site use the T09MCALL macro to call SEND:

```
LA    R01,CPTADT
ST    R01,CPTPARMS
T09MCALL SEND,PARM=CPTPARMS
```

However, a programmer can call the SEND interface stub directly:

```
LA    R01,CPTADT
ST    R01,CPTPARMS
LA    R01,CPTPARMS

L     R15,=V(T09FSEND)
BALR  R14,R15
```

Recommended ADT Parameters

The following table lists the recommend parameters to use with the SEND service. These parameters are set within the ADT control block.

For a complete list of optional parameters, see Complete Parameter List

Field Name	Description
ADTBUFFA	Set to address of user data area.
ADTBUFFL	Set to the length of the record in the ADTBUFFA field.
ADTTOKEN	Data transfer token, which identifies the session.
ADTVERS	Version should be set to 2. The T09MCALL macro automatically sets the correct version number in the ADT parameter list.

To use the LL SEND Option method you must move ADTTYPLL to the ADTOPTNS field in addition to setting the recommended fields above.

The following table contains the recommended **optional** parameters needed to use the Separator Character SEND Option method in addition to setting the previous recommended fields.

Field Name	Description
ADTOPTNS	Receiving method: set to ADTTYPSP.
ADTSEP#	Set to 1 or 2 for the number of separator characters.
ADTSEP1	First or only separator character.
ADTSEP2	Second separator character in a sequence of two.

When a caller specifies the Separator option type using setting ADTTYPSP, the Unicenter SOLVE:CPT SEND service appends the specified separator characters after the data packet before sending it to the remote.

LL SEND Option

LL SEND Option

In the LL SEND option, records are prefixed by a two-byte hexadecimal length field indicating the number of bytes to follow.

LL Example

The following is a hexadecimal example of a data stream where a two-byte LL length field containing x'000A' precedes the network data, where a hexadecimal example of a data stream that is passed to the SEND service as addressed by the ADTBUFFA field.

```
D4E8E3C5E2E3D9C5C3F2
```

The SEND service creates the following hexadecimal string before sending the packet to the remote. The LL header appears in **bold**.

000AD4E8E3C5E2E3D9C5C3F2

where:

000A	Length of the data record.
MYTESTREC2	The character data in the record.

Separator Character SEND Option

Separator Character SEND Option

In the separator character SEND option, the records are delimited by appending one or two separator characters after the data.

Note: The major limitation with the separator character SEND option is that the data may never contain a natural occurrence of the separator characters.

Double Separator Character Example

The following is a hexadecimal example of a data stream using x'0D0A' as a separator character, shown in **bold**. In this example, a hexadecimal example of a data stream that is passed to the SEND service as addressed by the ADTBUFFA field.

E3C5E2E3D9C5C3F1

The SEND service creates the following hexadecimal string before sending the packet to the remote. Two separator characters CRLF (carriage return) are appended to the end, shown in **bold**:

E3C5E2E3D9C5C3F1**0D0A**

where:

TESTREC1	The data record to send.
0D0A	CRLF terminates the data stream sent to the remote.

Single Separator Character Example

The following is a hexadecimal example of a data stream using x'FF' as a separator character which is in **bold**. Also, a data stream is passed to the SEND service as addressed by the ADTBUFFA field.

```
D4E8E3C5E2E3D9C5C3F2
```

The SEND service creates the following hexadecimal string before sending the packet to the remote. The single separator character x'FF' is appended to the end, shown in **bold**.

D4E8E3C5E2E3D9C5C3F2**FF**

where:

MYTESTREC2	The character data record to send.
FF	Terminates the data stream that is sent to the remote.

Usage Examples

There are three types of SEND calls that an application can use:

- Data SEND Example—Sends data without any changes or record indicators
- LL SEND Example—Data is prefixed with a LL length
- Separator Character SEND Example—Data is terminated by separator characters

Data SEND Example

In this example, the data is placed out in the network exactly as it was placed into the ADTBUFFA buffer.

It is up to the remote application to determine when it has received all the data of a particular record.

The token is loaded into the ADTTOKEN field from the ACM. The ADTBUFFA field contains the data buffer address. The ADTBUFFL contains the length of data to send across the network.

The application checks the ADTRTNCD return code field to determine the SEND service completion status.

Note: The statements related to the SEND service appear in **bold**.

```
*        Dsect's

         T09DACM MF=DSECT          Argument for Connection Management
         T09DADT MF=DSECT         Argument for SEND
*
*        Working storage
DFHEISTG DSECT
ACMARG   DS    XL(ACMLEN)          ACM for endpoint
SENDARG  DS    XL(ADTLEN)         Argument for SEND
CPTIOBUF DS    XL1024              CPT RECV/SEND BUFFER
*
label    DFHEIENT
*
         ...
*
          L    R9,ACMTOKEN         Load ACM Token
*
SEND     DS    0H
         USING ADT,SENDARG
*
         ST    R9,ADTTOKEN         Save connection token
         LA    R04,CPTIOBUF         LOAD I/O BUFFER ADDRESS
         ST    R04,ADTBUFFA        SAVE RECEIVE BUFFER ADDRESS
         LA    R05,L'MYMSG          LOAD I/O BUFFER LENGTH
         ST    R05,ADTBUFFL        SAVE RECEIVE BUFFER LENGTH
*
         LA    R01,SENDARG         Point to control block
         ST    R01,CPTPARMS        Store address of control block
         T09MCALL SEND,PARM=CPTPARMS
         LTR   R15,R15             Test Return Code
         BZ    SENDOK               Good==>SENDOK
*                                   Bad...
*        SEND error from ADT
*
         L     R3,ADTRTNCD          Load SEND Return Code
         L     R4,ADTDGNCD          Load SEND Diagnostic Code
          ...
*
SENDOK   DS    0H
```

LL SEND Example

In this example, the data is sent out on to the network prefixed by a two-byte length field followed by the data from the ADTBUFFA buffer of size ADTBUFFL.

The remote application knows how many bytes it has to read after it examines the first two bytes of LL length sent in the data packet.

The token is loaded into the ADTTOKEN field from the ACM. The ADTBUFFA field contains the data buffer address. The ADTBUFFL contains the caller's length of data to send across the network. The ADTTYPLL flag directs the SEND process to convert the ADTBUFFL value to a two-byte field and place it out on the network before the data in the ADTBUFFA buffer.

The application checks the ADTRTNCD return code field to determine the SEND service completion status.

Note: The statements related to the SEND service appear in **bold**.

```
*        Dsect's

         T09DACM MF=DSECT          Argument for Connection Management
         T09DADT MF=DSECT         Argument for SEND
*
*        Working storage
DFHEISTG DSECT
ACMARG   DS    XL(ACMLEN)          ACM for endpoint
SENDARG  DS    XL(ADTLEN)         Argument for SEND
CPTIOBUF DS    XL1024              CPT RECV/SEND BUFFER
*
label    DFHEIENT
*
         ...
*
          L    R9,ACMTOKEN         Load ACM Token
*
SEND     DS    0H
         USING ADT,SENDARG
*
         ST    R9,ADTTOKEN         Save connection token
         MVI   ADTOPTCD1,ADTTYPLL  Set LL TYPE SEND
         LA    R04,CPTIOBUF         LOAD I/O BUFFER ADDRESS
         ST    R04,ADTBUFFA        SAVE RECEIVE BUFFER ADDRESS
         LA    R05,L'MYMSG          LOAD I/O BUFFER LENGTH
         ST    R05,ADTBUFFL        SAVE RECEIVE BUFFER LENGTH
*
         LA    R01,SENDARG         Point to control block
         ST    R01,CPTPARMS        Store address of control block
         T09MCALL SEND,PARM=CPTPARMS
         LTR   R15,R15             Test Return Code
         BZ    SENDOK               Good==>SENDOK
*                                   Bad...
*        SEND error from ADT
*
         L     R3,ADTRTNCD          Load SEND Return Code
         L     R4,ADTDGNCD          Load SEND Diagnostic Code
          ...
*
SENDOK   DS    0H
```

Separator Character SEND Example

In this example, the ADTBUFFA data is sent out on the network post fixed by two bytes containing carriage return (x'0D') and line feed (x'0A').

The remote application knows it has to read all the bytes in a record when it finds two bytes in the data stream that match the ADTSEP1 and ADTSEP2 fields.

The token is loaded into the ADTTOKEN field from the ACM. The ADTBUFFA field contains the data buffer address. The ADTBUFFL contains the caller's length of data to send across the network. The ADTTYPSP flag along with the value two set in the ADTSEP# field directs the SEND process to send the ADTBUFFA buffer data followed by the ADTSEP1 and ADTSEP2 character values.

The application checks the ADTRTNCD return code field to determine the SEND service completion status.

Note: The statements related to the SEND service appear in **bold**.

```
*        Dsect's

         T09DACM MF=DSECT          Argument for Connection Management
         T09DADT MF=DSECT         Argument for SEND
*
*        Working storage
DFHEISTG DSECT
ACMARG   DS    XL(ACMLEN)          ACM for endpoint
SENDARG  DS    XL(ADTLEN)         Argument for SEND
CPTIOBUF DS    XL1024              CPT RECV/SEND BUFFER
*
label    DFHEIENT
*
         ...
*
          L    R9,ACMTOKEN         Load ACM Token
*
SEND     DS    0H
         USING ADT,SENDARG
*
         ST    R9,ADTTOKEN         Save connection token
         MVI   ADTOPTCD1,ADTTYPSP  Set Separator character TYPE SEND
         MVI   ADTSEP#,2           2 Separator Characters
         MVI   ADTSEP1,X'0D'       CR is 1st Separator Character
         MVI   ADTSEP1,X'0A'       LF is 2nd Separator Character
         LA    R04,CPTIOBUF         LOAD I/O BUFFER ADDRESS
         ST    R04,ADTBUFFA        SAVE RECEIVE BUFFER ADDRESS
         LA    R05,L'MYMSG          LOAD I/O BUFFER LENGTH
         ST    R05,ADTBUFFL        SAVE RECEIVE BUFFER LENGTH
*
         LA    R01,SENDARG         Point to control block
         ST    R01,CPTPARMS        Store address of control block
         T09MCALL SEND,PARM=CPTPARMS
         LTR   R15,R15             Test Return Code
         BZ    SENDOK               Good==>SENDOK
*                                   Bad...
*        SEND error from ADT
*
         L     R3,ADTRTNCD          Load SEND Return Code
         L     R4,ADTDGNCD          Load SEND Diagnostic Code
```

```
*          ...
SENDOK   DS    0H
```

Parameter Values Returned in the ADT

After the SEND call returns control to your application program, the following fields are propagated with the results of the SEND service call. These updated values are passed back to the application in the ADT control block.

Parameters	Description
ADTBUFFL	The number of user data bytes actually sent.
ADTDGNCD	Diagnostic code.
ADTRTNCD	Return code.

Assembler DSECTs

Sample Assembler DSECTs are provided in the distributed software and are available to you in ***cpthlq***.T09MAC. Variable field names contained in the distributed samples and the examples in this guide refer to these DSECTs.

T09DADT — Assembler DSECT name for the ADT. For detailed information and a sample copy of the Assembler DSECT, see the ADT: Argument for Data Transfer Used by RECIEVE, SEND, RECVFROM, and SENDTO Services Service section in appendix “Control Block Layouts.”

All Assembler constants that apply to ADT calls are imbedded in the ADT DSECT sample.

Sample Programs

Sample Assembler source code is available in the distributed software in the ***cpthlq***.T09SAMP library. You should be able to find a sample that matches your programming requirement. For complete details on the function a sample program provides, see the program descriptions in the "Unicenter SOLVE:CPT API Services" chapter and the descriptions of the sample members listed below.

Name	Description
T09PACL1	Client Application: Sends typed in data to the server waiting for the information to be echoed back from the server.
T09PACL2	Client Application: Sends an internal message using the FULL, SEP or LL to be echoed back by the server.
T09PASV1	TCP Server 1 program: A single-threaded server using a Listen API call.
T09PASV2	TCP Server 2 program: A multithreaded server using the LISTEN tool.
T09PASV5	TCP Server 5 program: Like SV2 and uses the Select tool.

Completion Information

The SEND service sends normal data as output through a Unicenter SOLVE:CPT connection. The data may be 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 remote 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 call 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. Typically, data generated by the application is forwarded when it is sent in a continuous flow.

The SEND service completes after it copies data from the ADTBUFFA buffer into the internal TCP/IP data buffers for data transfer across the network.

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 SEND requests. Multiple SEND requests do not present a problem. A large buffer value can waste application storage.

The buffer size value is specified during connection initialization and can be modified on return. An application that is dependent on the buffer size value 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 ADTVERS version number indicates the Unicenter SOLVE:CPT release level in which this user application program is written. This required field must be set to ADTVERSN and is validated by the SEND service before processing the request.

The ADTFUNC function code indicates the Unicenter SOLVE: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 ensures 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 ADTBUFFL field specifies the data buffer. 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 indicating the number of bytes processed.

Return Codes

The SEND service returns a code in registers R15 and R0 indicating the results of the execution. These values are in the ADTRTNCD (R15) and ADTDGNCD (R0).

DSECT T09DRTCD contains equates and descriptions for the possible return codes. T09DRTCD is available in the distributed software in ***cpthlq***.T09MAC. See the appendix “Return Codes” for a sample copy of the T09DRTCD DSECT.

This table describes the SEND service return codes.

Decimal	Hex	Diagnostic Code	Variable	Description
0	0	No	CPTIRCOK	Request completed successfully.
6	6	35=EWOULDBLOCK	CPTWBLCK	The SEND request can send only what is available for the SEND buffers. The ACMMSEND value will hold the largest possible block of data that can be sent.
17	11	No	CPTEVERS	Control block version number not supported.
20	14	No	CPTETOKN	Specified data transfer token is invalid.
21	15	No	CPTEBUFF	Buffer address and/or length invalid.
28	1C	No	CPTESEP#	Bad value in ADTSEP# field
29	1D	No	CPTEOPTN	Caller has requested both ADTTYPSP and ADTTYPLL.
31	1F	No	CPTEFRMT	Other Socket Call Parameter List format or specification error.
34	22	No	CPTENAPI	API not fully available; retry.
40	28	Yes	CPTETERM	Environment is being terminated.
47	2F	Yes	CPTEENVR	Other transport layer

Decimal	Hex	Diagnostic Code	Variable	Description
				environmental condition.
65	41	Yes	CPTERLSE	Orderly release of remote connection request.
68	44	Yes	CPTEDISC	Remote connection not available or aborted.
72	48	Yes	CPTEPRGE	Remote connection environment terminating.
79	4F	Yes	CPTEINTG	Other transport layer connection/data integrity error.
143	8F	Yes	CPTEPROC	Procedural error.
254	FE	Is abend code	CPTABEND	Abnormal termination. **Note:** The diagnostic code is the abnormal termination code, which is normally a CICS abend code, but can also be in the "Abend Codes" chapter of the *Message Guide*.
255	FF	No	CPTEOTHR	Other error.

Usage Notes

The SEND service sends normal data as output through a Unicenter SOLVE:CPT connection. The data may be 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 not 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 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 ADTVERS version number indicates the Unicenter SOLVE:CPT release level in which this user application program is written. This required field must be set to ADTVERSN and is validated by the SEND service before processing the request.

The ADTFUNC function code indicates the Unicenter SOLVE: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 ADTTOKEN token 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 ADTOPTNS field specifies SEND processing control options and provides a mechanism for event notification on return to the application program.

Complete Parameter List

ADTBUFFA

User data address. ADTBUFFA indicates the address of user data to send 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 Unicenter SOLVE:CPT or the transport provider. The storage area can be aligned on any boundary convenient for the application program.

Default: None.

ADTBUFFL

User data length. ADTBUFFL indicates the length, in bytes, of user data in the storage area 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: None.

ADTDGNCD

Diagnostic code. Indicates the diagnostic code set by the service request. This value generally indicates a transport provider return code.

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 program.

Default: None.

ADTLADDR

Used only by the UDP calls RCVFROM and SENDTO.

For TCP connections, this parameter is set in the equivalent ACM field.

ADTLNAME

Used only by the UDP calls RCVFROM and SENDTO.

For TCP connections, this parameter is set in the equivalent ACM field.

ADTLPORT

Used only by the UDP calls RCVFROM and SENDTO.

For TCP connections, this parameter is set in the equivalent ACM field.

ADTMRECV

Used only by the UDP calls RCVFROM and SENDTO.

For TCP connections, this parameter is set in the equivalent ACM field.

ADTMSEND	Used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
ADTMSOCK	Has no meaning for TCP connections, since the maximum number of sockets is set at connection establishment time through the ACMMSOCK field.
ADTNSLCT	Number of entries in the SELECT vector. Not used by the SEND service.
ADTOPTNS	Specifies data transfer options. These are the ADT options that apply to TCP data transfer requests:
ADTOPCD2	Options for byte 2.

ADTFVLST	Currently for internal use only.
ADTNOSTP	Do not strip record delimiter sequence. Not used by the SEND service.
ADTNWAIT	Do not wait for completion. Not used by the SEND service.
ADTNOQUE	Do not QUEUE API RECEIVES. Not used by the SEND service.

ADTOPCD1	Options for byte 1.	
	ADTBLCKS	This option was disabled with the CPT 6.1 API conversion. This option is ignored.
	ADTFDNR.	Do DNR name resolution (not used by the SEND service).
	ADTNBLKR	Do not block on a call to the RECEIVE service. Not used by the SEND service.
	ADTTMPRT	Timed partial record RECEIVE. Not used by the SEND service.
	ADTTMRCV	Timed full record RECEIVE. Not used by the SEND service.
	ADTTYPLL	LL type SEND. See LL SEND for an example of how ADTTYPLL works. These fields (along with other required ADT fields) are used to request a LL type SEND call: ADTOPTNS = ADTTYPLL
	ADTTYPSP	SEP type SEND. See Separator Character SEND for an example of how ADTTYPSP works. These files (along with other required ADT fields) are used to request a SEP type SEND call: ADTOPTNS = ADTTYPSP ADTSEP# = 1 OR 2 ADTSEP1 = character ADTSEP2 = character if ADTSEP# = 2
	It is an error to combine these SEND service options: ADTTYPLL ADTTYPSP **Note:** An invalid combination will result in CPTEOPTN being returned in ADTRTNCD. Default: None.	
ADTQRECV	Used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field. The only valid value is one.	

ADTQSEND.	Used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field. The only valid value is one.
ADTRADDR	Used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
ADTRNAME	Used only by the UDP calls RCVFROM and SENDTO.
ADTRPORT	Used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
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: None.
ADTSEP#	Number of separator characters for option ADTTYPSP (0 < ADTSEP# < 3). If ADTSEP# is not equal to 1 or 2, CPTESEP# is 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.
ADTSTAT	Specifies statistics logging options for the application program. ADTSCONN — Specifies that messages be generated on the initial connection of a session. ADTSTERM — Specifies that messages be generated on terminating an established connection. Default: Zero, no statistics logging.
ADTTIMEO	RECEIVE timeout value. Not used by the SEND service. Default: None.

ADTTOKEN	It specifies the token, which represents a TCP connection. If the ADT is passed in a call to either the RECEIVE or SEND service, it must be a token representing a previously established TCP connection, using 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: None.
ADTTRACE	Note that the tracing functionality has moved in version 6 of Unicenter SOLVE:CPT. A greatly enhanced tracing capability is now available via the TCPEEP tracing command. See the *Administrator Guide* for more detail. These tracing fields remain only for downward compatibility purposes and are ignored.

ACMTNTRY	ACMTTERM	ACMTTPL
ACMTARGS	ACMTPASS	ACMTRLSE
ACMTRECV	ACMTCLSE	ACMTSTOR
ACMTSEND	ACMTTERR	ACMTCLTD

ADTUCNTX	Used only by the UDP calls RCVFROM and SENDTO. For TCP connections, this parameter is set in the equivalent ACM field.
ADTVECTR	Address of the SELECT vector. Not used by the SEND service.
ADTVERS	Version. Indicates the Unicenter SOLVE:CPT version number of the argument used by the calling program. Must be set to a binary two for this release of Unicenter SOLVE:CPT. Default: None.

Chapter

11 SENDTO Service

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:

- Establishes a UDP client endpoint represented by a new token and sends a datagram to a remote UDP server.

  This function is indicated to the SENDTO service by passing an ADTTOKEN equal to zero. SENDTO then creates all the internal control blocks and the SENDTO buffer queue. Even though the RCVFROM buffer queue is not 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 Unicenter SOLVE:CPT control blocks at endpoint creation time. On return from the SENDTO service, ADTTOKEN contains the token value to pass to subsequent SENDTO and RCVFROM service calls.

- Sends 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. The other Unicenter SOLVE:CPT service calls GIVE, TAKE, and TRANSLATE are available to UDP applications.

This chapter discusses these topics:

- Call Syntax—Shows sample syntax for the SENDTO service call
- Recommended ADT Parameters—Lists the parameters normally used and recommended for the SENDTO service call
- Usage Example—Provides a sample program shell for using the SENDTO service call
- Parameter Values Returned in the ADT—List the fields that are updated in the ADT control block upon return from the SENDTO service call
- Assembler DSECTs—Provides a list and information about the distributed Assembler DSECTs that are used by the SENDTO service call
- Sample Programs—Lists and describes the distributed sample Assembler programs that use the SENDTO service call
- Network Considerations—Reviews network-related issues that may influence your environment
- Return Codes—Lists the return codes that can apply to the SENDTO service call
- Complete Parameter List—Provides a complete list of the parameters and their options for the SENDTO service call

Call Syntax

Computer Associates recommends that a site use the T09MCALL macro to call SENDTO:

```
LA    R01,CPTADT
ST    R01,CPTPARMS
T09MCALL SENDTO,PARM=CPTPARMS
```

However, a programmer can call the SENDTO interface stub directly:

```
LA    R01,CPTADT
ST    R01,CPTPARMS
LA    R01,CPTPARMS
L      R15,=V(T09FSNTO)
BALR   R14,R15
```

Recommended ADT Parameters

The following table lists the recommended parameters for use with the SENDTO service. These parameters are set within the ADT control block.

See Assembler DSECTs for sample information.

For a complete list of optional parameters, see Complete Parameter List.

Parameter	Description
ADTBUFFA	Set to address of user data area.
ADTBUFFL	Set to the maximum expected length of any record.
ADTRADDR	If ADTRNAME, is not used, Remote IP Host Address in hexadecimal.
ADTRPORT	Remote Well-Known Service Port.
ADTRNAME	Remote IP host name, mutually exclusive with ADTRADDR.
ADTTOKEN	Data transfer token, set to zero (0) if initial call.
ADTVERS	Version number should be set to 2. The T09MCALL macro automatically sets the correct version number in the ADT parameter list.

Usage Example

In this example, a subset of the actual statements required is shown to emphasize the use of a SENDTO call. In the example, a message is sent to a remote host. The ADTRTNCD return code is checked to determine SENDTO service completion status.

Note: The statements needed for the SENDTO service appear in **bold**.

```
*         Dsect's

          T09DACM MF=DSECT          Argument for Connection Management
          T09DADT MF=DSECT          Argument for SENDTO
*
*         Working storage
DFHEISTG DSECT
SNDTOARG DS     XL(ADTLEN)          Argument for SENDTO
*
CPTSBUFF DS     XL80                 Data buffer for Message
CPTLENG  DS     H                    LENGTH OF INPUT MESSAGE
*
label    DFHEIENT
*
*         CPT Transfer
*
SENDTO   DS      0H
          USING  ADT,SNDTOARG
*
          MVC    ADTRNAME(15),=C'123.234.105.199' Remote IP ADDR
          MVC    ADTRPORT,=AL2(1980) SELECT THE PORT
          LA     R04,CPTSBUFF         LOAD SEND BUFFER ADDRESS
          ST     R04,ADTBUFFA        SAVE SOURCE ADDRESS
          LH     R05,CPTLENG          LOAD LENGTH OF SOURCE
          ST     R05,ADTBUFFL        SAVE SOURCE LENGTH
*
          LA     R01,SNDTOARG        Point to control block
          ST     R01,CPTPARMS        Store address of control block
          T09MCALL SENDTO,PARM=CPTPARMS
          LTR    R15,R15             Test Return Code
          BZ     CALLRCFR             Good,
*
*          SENDTO error
*
          L      R3,ADTRTNCD          Load ADT Return Code
          L      R4,ADTDGNCD          Load ADT Diagnostic Code
```

Parameter Values Returned in the ADT

After the SENDTO call returns control to your application program, the following fields are propagated with valid established connection information. These updated values are passed back to the application in the ADT control block.

Parameters	Description
ADTLADDR	Local IP Host Address.
ADTBUFFL	The number of user data bytes actually sent.
ADTDGNCD	Diagnostic Code.
ADTLNAME	Local IP Host Name.
ADTLPORT	Client Application Port.
ADTMRECV	API receive buffer size.
ADTMSEND	API send buffer size.
ADTQRECV	API receive queue size, set to one.
ADTQSEND	API send queue size, set to one.
ADTRADDR	Remote IP Host Address.
ADTRTNCD	Return Code.
ADTTOKEN	Token—Connection or endpoint.

Assembler DSECTs

Sample Assembler DSECTs are provided in the distributed software and are available to you in ***cpthlq***.T09MAC. Variable field names contained in the distributed samples and the examples in this guide refer to these DSECTs.

T09DADT	Assembler DSECT name for the ADT. For detailed information and a sample copy of the Assembler DSECT, see the ADT: Argument for Data Transfer Used by RECIEVE, SEND, RECVFROM, and SENDTO Services Service section in appendix "Control Block Layouts."

All Assembler constants that apply to ADT calls are imbedded in the ADT DSECT sample.

Sample Programs

Sample Assembler source code is provided for your use. You should be able to find a sample that matches your programming requirement. For more complete details on what function a sample program provides, see the program descriptions in the "Unicenter SOLVE:CPT API Services" chapter and the comments at the beginning of the sample members listed below. These sample program members are available in the distributed software in the ***cpthlq***.T09SAMP library.

Name	Description
T09PACLU	Sample UDP client.
T09PASVU	Sample UDP server.

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.

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.
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 is only returned if ADTFDNR is specified in ADTOPCD1.

Return Codes

The SENDTO service returns a code in registers R15 and R0 indicating the results of the execution. These values are in the ADTRTNCD (R15) and ADTDGNCD (R0).

DSECT T09DRTCD contains equates and descriptions for the possible return codes. T09DRTCD is available in the distributed software in ***cpthlq***.T09MAC. See the appendix “Return Codes” for a sample copy of the T09DRTCD DSECT.

The following table lists the return codes that can apply to the SENDTO call.

Decimal	Hex	Diagnostic Code	Variable	Description
0	0	No	CPTIRCOK	Request completed successfully.
6	6	Yes	CPTWBLCK	Non-blocking call to the RCVFROM service.
17	11	No	CPTEVERS	Control block version number not supported.
18	12	Yes	CPTECONN	Required Parameter not passed. For example, host, port, …
19	13	No	CPTEPROT	Specified protocol not supported.
20	14	No	CPTETOKN	Specified data transfer token is invalid.
21	15	No	CPTEBUFF	Buffer address and/or length invalid.
31	1F	No	CPTEFRMT	Other Socket Call Parameter List format or specification error.
34	22	No	CPTENAPI	API not fully available; retry.
40	28	Yes	CPTETERM	Environment is being terminated.
47	2F	Yes	CPTEENVR	Other transport layer environmental condition.
65	41	Yes	CPTERLSE	Orderly release of remote connection request.
68	44	Yes	CPTEDISC	Remote connection not available or aborted.

Decimal	Hex	Diagnostic Code	Variable	Description
72	48	Yes	CPTEPRGE	Remote connection environment terminating.
79	4F	Yes	CPTEINTG	Other transport layer connection/data integrity error.
143	8F	Yes	CPTEPROC	Procedural error.
254	FE	Is abend code	CPTABEND	Abnormal termination. Note that the diagnostic code is the abnormal termination code, which is normally a CICS abend code, but can also be in the “Abend Codes” chapter of the *Message Guide*.
255	FF	No	CPTEOTHR	Other error.

Complete Parameter List

ADTBUFFA

User data address. Indicates the storage address from which the UDP datagram is 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 Unicenter SOLVE:CPT or the transport provider. The storage area can be aligned on any boundary convenient for the application program.

Default None.

ADTBUFFL

Specifies the length in bytes of the ADTBUFFA field.

On return to the caller, ADTBUFFL reflects the number of bytes actually sent (generally the number requested).

It is an error to call the SENDTO service with an ADTBUFFL of zero.

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.

ADTDGNCD

Diagnostic code. Indicates the diagnostics code set by the SENDTO service. This value generally indicates a transport provider return code.

Default: None.

ADTLADDR

Local IP host address. Indicates the local host internet address. The local host internet address is returned to the caller of the SENDTO service.

This field is an unsigned four-byte integer value.

Default: None.

ADTLNAME

Local IP host name. Indicates the local host internet name. The local host internet name is returned to the caller of the SENDTO service.

This field is a 255-byte character string that is padded with blanks.

Default: None.

ADTLPORT	Local well-known service port. Indicates the local transport layer port from which the calling application will be sending (SENDTO) UDP datagrams. 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.
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.
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. Default: 1024.
ADTNSLCT	Number of entries in the SELECT vector. Not used by the SENDTO service.

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.
ADTTMRCV	This option allows the caller to wait up to specified amount of time for a datagram. It must be used with the ADTNBLKR option, and ADTTIMEO must be specified. This option is not used by the SENDTO service.

These options can be toggled on every UDP data transfer call even if the caller is using the same token.

Default: None.

ADTQRECV	API receive queue size (used when ADTTOKEN=0). You should only specify one. Adding extra buffers wastes storage and does not improve performance. Default: One.
ADTQSEND	API send queue size (used when ADTTOKEN=0). You should only specify one. Adding extra buffers wastes storage and does not improve performance. Default: One.
ADTRADDR	Remote IP host address. Specifies the remote host internet address destination for the datagram being processed by the SENDTO service. This field is an unsigned four-byte integer value. Default: None

ADTRNAME	Remote IP host name. Indicates the remote host internet name. 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. This field is a 255-byte character string that is padded with blanks. Default: None.
ADTRPORT	Remote port. Indicates the remote 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.
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: 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.
ADTSLCTD	Number of tokens selected. Not used by the SENDTO service.
ADTSRVCE	This field remains only for downward compatibility purposes and is ignored. This field is no longer supported in Version 6 of CPT.

ADTSTAT — Specifies statistics logging options for the application program.

ADTSCONN	Specifies that messages be generated on the closing of a UDP token. These messages are generated by the Unicenter SOLVE:CPT CLOSE service.
ADTSTERM	Specifies that messages be generated on terminating an established connection. These messages are generated by the Unicenter SOLVE:CPT CLOSE service.

Default: None, no statistics logging.

ADTTIMEO — RECEIVE timeout value. Not used by the SENDTO service

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: None.

ADTTRACE — Note that the tracing functionality has moved in Version 6 of Unicenter SOLVE:CPT. A greatly enhanced tracing capability is now available via the TCPEEP tracing command. See the *Administrator Guide* for more detail. These tracing fields remain only for downward compatibility purposes and are ignored.

ADTTNTRY	ADTTTERM	ADTTTPL
ADTTARGS	ADTTPASS	ADTTRLSE
ADTTRECV	ADTTCLSE	ADTTSTOR
ADTTSEND	ADTTTERR	ADTTCLTD

ADTUCNTX — One word of user context. Specifies one arbitrary word of user context to associate with the endpoint. The information provided is not interpreted by Unicenter SOLVE:CPT, and is saved with other endpoint information.

Default: None, No user context.

ADTVECTR Address of the SELECT vector. Not used by the SENDTO service.

ADTVERS Required. Version. Indicates the Unicenter SOLVE:CPT Version number of the argument used by the calling program.

Must be set to binary two for this release of Unicenter SOLVE:CPT.

Default: None.

Chapter

12 TAKE Service

Acquires ownership of a connection and associated internal Unicenter SOLVE:CPT resources. You must use the TAKE service call to guarantee proper passing of a connection from another transaction.

To invoke the TAKE service, a user application must first build an AFM (Argument for Facility Management) and then 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 that indicates the success or failure of the request.

This chapter discusses the following topics:

- Call Syntax—Sample syntax for the TAKE service call
- Recommended AFM Parameters—Lists the parameters normally used and recommended for the TAKE service call
- Usage Example—Provides a sample program shell using the TAKE service call
- Parameter Values Returned in the AFM—Lists fields that are updated in the AFM control block upon return from the TAKE service call
- Assembler DSECTs—Lists information about the distributed sample Assembler DSECTs used by the TAKE service call
- Sample Programs—Lists sample Assembler programs that use the TAKE service call
- Completion Information—Describes the expected results at completion of a TAKE service call.
- Return Codes—Gives a list of return codes that can apply to the TAKE service call
- Usage Notes—Provides miscellaneous notes about TAKE service call
- Complete Parameter List—Gives a complete list of all the parameters and their options for the TAKE service call

Call Syntax

We recommend that a site use the T09MCALL macro to call TAKE:

```
LA    R01,CPTAFM
ST    R01,CPTPARMS
T09MCALL TAKE,PARM=CPTPARMS
```

However, a programmer can call the TAKE interface stub directly:

```
LA    R01,CPTAFM
ST    R01,CPTPARMS
LA    R01,CPTPARMS

L      R15,=V(T09FTAKE)
BALR   R14,R15
```

Recommended AFM Parameters

The following table lists the recommended parameters to use with the TAKE service. These parameters are set within the argument list of the AFM control block. See the Assembler DSECTs for more information.

For a complete list of optional parameters, see Complete Parameter List.

Parameters	Description
AFMVERS	Version number should be set to 2. The T09MCALL macro automatically sets the correct version number in the AFM parameter list.
AFMTOKEN	Required session token specifies which session that the current task wishes to take control over.

Usage Example

In this example, a subset of the actual statements required is shown to emphasize the use of a TAKE call. In the example, a data processing transaction retrieves the ACM, copies ACMTOKEN to the AFMTOKEN field, issues the TAKE service to take control of the session represented in the ACMTOKEN field. The return code is checked to determine TAKE service completion status.

Note: The statements needed for the TAKE service appear in **bold**.

```
*        Dsect's

         T09DACM MF=DSECT          Argument for Connection Management
         T09DAFM MF=DSECT          Argument for TAKE
*
*        Working storage
DFHEISTG DSECT
ACMARG   DS     XL(ACMLEN)         ACM for endpoint
TAKEARG  DS     XL(AFMLEN)         Argument for TAKE
RTRVLEN  DS     H
*
label    DFHEIENT
*
         EXEC  CICS RETRIEVE                                    X
               NOHANDLE                                         X
               SET     (ACMARG)                                 X
               LENGTH  (RTRVLEN)
*
          L     R9,ACMTOKEN        Load ACM Token
*
TAKE     DS     0H
         USING AFM,TAKEARG
*
         ST     R9,AFMTOKEN        Save connection token
*
         LA     R01,TAKEARG       Point to control block
         ST     R01,CPTPARMS       Store address of control block
         T09MCALL TAKE,PARM=CPTPARMS
         LTR    R15,R15            Test Return Code
         BZ     TAKEOK             Good==>TAKEOK
*                                  Bad...
*        TAKE error from AFM
*
         L      R3,AFMRTNCD        Load TAKE Return Code
         L      R4,AFMDGNCD        Load TAKE Diagnostic Code
          ...
*
TAKEOK   DS     0H
*
          ...
```

Parameter Values Returned in the AFM

After the TAKE call returns control to your application program, the following fields are propagated with the call's resultant information. These updated values are passed back to the application in the AFM control block.

Parameters	Description
AFMDGNCD	Diagnostic code.
AFMRTNCD	Return code.

Assembler DSECTs

Sample Assembler DSECTs are provided in the distributed software and are available to you in ***cpthlq***.T09MAC. Variable field names contained in the distributed samples and the examples in this guide refer to these DSECTs.

T09DAFM	Assembler DSECT name for the AFM. For detailed information and a sample copy of the Assembler DSECT, see AFM: Argument for Facility Management Used by the GIVE and TAKE Services Service section in appendix "Control Block Layouts."

All Assembler constants that apply to AFM calls are imbedded in the AFM DSECT sample.

Sample Programs

Sample Assembler source code is available in the distributed software in the ***cpthlq***.T09SAMP library. You should be able to find a sample that matches your programming requirement. For complete details on the function a sample program provides, see the program descriptions in the "Unicenter SOLVE:CPT API Services" chapter and the descriptions of the sample members listed below.

Name	Description
T09PASV2	TCP Server 2 program is a multithreaded server using the Listen tool.
T09PASV5	TCP Server 5 program is spawned by an inbound connection from the T09MLSTN tool. It uses the SELECT tool to handle RECEIVE calls when there is no available data.

Completion Information

The TAKE service completes normally when the task takes control of the session associated with the token passed from the AFMTOKEN field.

On normal return to the application program, the general return code in AFMRTNCD is set to zero (CPTIRCOK).

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 zero indicate the nature of the failure

Return Codes

The TAKE service returns a code in registers R15 and R0 indicating the results of the execution. These values are in the AFMRTNCD (R15) and AFMDGNCD (R0).

DSECT T09DRTCD contains equates and descriptions for the possible return codes. T09DRTCD is available in the distributed software in ***cpthlq***.T09MAC. See the appendix "Return Codes" for a sample copy of the T09DRTCD DSECT.

The following table lists the return codes that can apply to the TAKE call.

Decimal	Hex	Diagnostic Code	Variable	Description
0	0	No	CPTIRCOK	Request completed successfully.
17	11	No	CPTEVRSN	Control block version number not supported.
20	14	No	CPTETOKN	Specified data transfer token is invalid.
34	22	No	CPTENAPI	API not fully available; retry.
40	28	No	CPTETERM	TCPIP is terminating.
254	FE	Is abend code	CPTABEND	Abnormal termination. **Note:** The diagnostic code is the abnormal termination code, which is normally a CICS abend code, but can also be in the "Abend Codes" chapter of the *Message Guide*.
255	FF	No	CPTEOTHR	Other error.

Usage Notes

The TAKE service acquires ownership of a connection from one task to another. This service is non-blocking and does not affect any pending transport provider data transfer requests. The association established by the TAKE service lets the Unicenter SOLVE: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 Unicenter SOLVE:CPT with resource management capabilities.

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

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

The GIVE and TAKE services are all about proper resource cleanup. For a Unicenter SOLVE:CPT token (connection) to be properly passed to another transaction, it must first be GIVEn to release ownership. The receiving transaction must then TAKE the connection before using the connection.

Important! *As noted in the GIVE service, if a transaction does not GIVE the token before it performs an EXEC CICS RETURN then the CICS TRUE end of task exit will cleanup all resources including closing down the connection. Therefore, if you have not GIVEn your token, the next transaction will not be able to use the connection because it will be gone; already be closed. So, a TAKE will fail. As with the GIVE, as the next user of the connection you must use a TAKE to establish yourself as the owner. By doing the TAKE, you avoid possibilities of your connection being accidentally cleaned up.*

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 Unicenter SOLVE:CPT task termination exit) if the server application terminates. However, if the data transfer transaction terminates without issuing an explicit close (Unicenter SOLVE:CPT CLOSE service) an implicit close is scheduled and resource management is handled by the Unicenter SOLVE: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 gets control of the connection and associated resources. We recommend 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 AFMVERS version number indicates the Unicenter SOLVE:CPT release level in which this user application program is written. This required field must be set to AFMVERSM (2) and is validated by the TAKE service before processing the request.

The AFMFUNC function code indicates the Unicenter SOLVE: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.

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.

Complete Parameter List

AFMCOMMA	Reserved for future use.
AFMCOMML	Reserved for future use.
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. 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.
AFMOPCD1	Option byte. AFMOPSEL — Informs the GIVE service to pass this token to the SELECT TOOL AFMOPCOM — Reserved for future use. Default: None.
AFMMSOCK	Unused parameter.
AFMNTRAN	Transaction ID. Not used by the TAKE service
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.
AFMTOKEN	Required session token specifies which session that the current task wishes to take control over. Default: None.
AFMVERS	Required version number. Indicates the Unicenter SOLVE:CPT version number of the argument list used by the calling program. Must be set to a binary two for this release of Unicenter SOLVE:CPT. Default: None.

Chapter 13

TRANSLATE Service

The TRANSLATE service translates data between EBCDIC and ASCII character sets. Unicenter SOLVE: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 chapter discusses the following topics:

- Call Syntax—Shows syntax of the TRANSLATE call
- Recommended AXL Parameters—Lists the parameters normally used and recommended for the TRANSLATE service call
- Usage Example—Provides a sample program shell using the TRANSLATE service call
- Custom Translation Table Usage Notes—Provides information on how to configure and use a custom translation table for your environment
- Parameter Values Returned in the AXL—Lists the fields that are updated in the AXL control block upon return from the TRANSLATE service call
- Assembler DSECTs—Lists information about the distributed sample Assembler DSECTs used by the TRANSLATE service call and available in *cpthlq*.T09SAMP
- Sample Programs—Lists and describes the distributed sample Assembler programs that use the TRANSLATE service call along with other service calls
- Complete Parameter List—Lists all of the parameters and options of those parameters for the TRANSLATE service call
- Return Codes—Lists the return codes that can apply to the TRANSLATE service call

- Usage Notes—Provides miscellaneous notes about the TRANSLATE service call
- Complete Parameter List—Lists all of the parameters and options of those parameters for the TRANSLATE service call

Call Syntax

We recommend that a site use the T09MCALL macro to call TRANSLATE:

```
LA    R01,CPTAXL
ST    R01,CPTPARMS
T09MCALL TRANS,PARM=CPTPARMS
```

However, a programmer can call the TRANSLATE interface stub directly:

```
LA    R01,CPTAXL
ST    R01,CPTPARMS
LA    R01,CPTPARMS

L      R15,=V(T09FXLAT)
BALR   R14,R15
```

Recommended AXL Parameters

The following table lists the recommended parameters to use with the TRANSLATE service. These parameters are set within the AXL control block. See Assembler DSECTs for sample information.

For a complete list of optional parameters, see the Complete Parameter List.

Parameter	Description
AXLLTYPE	Specify translation type or direction, set to: AXLTATOE Indicates ASCII to EBCDIC translation. AXLTETOA Indicates EBCDIC to ASCII translation.
AXLSADDR	Set to buffer address of data to be translated.
AXLSLENG	Set to length of data to be translated.
AXLTABLE	Optionally change translation table by setting this field to the address of user translation table.
AXLTOKEN	Connection or endpoint token.
AXLVERS	The version should be set to two (2).

Usage Example

In these examples, subsets of the actual statements required is shown to emphasize the use of a TRANSLATE call. For a reference to a more complete sample, see Sample Programs.

Two typical examples are shown:

- Inbound Translation Example—Translation from ASCII to EBCDIC
- Outbound Translation Example—Translation from EBCDIC to ASCII

Inbound Translation Example

In this example, the application has received data from a remote ASCII host. It sets flag AXLTATOE to translate the network data from ASCII to EBCDIC. The token AXLTOKEN, data buffer address AXLSADDR, and length to translate, AXLSLENG are set in the AXL. The default translation mode of SBCS is selected.

The application checks the AXLRTNCD return code to determine the TRANSLATE service completion status.

Note: Relevant parameters in the example are in **bold**:

```
*        Dsect's

         T09DACM MF=DSECT          Argument for Connection Management
         T09DAXL MF=DSECT          Argument for Translate
*        Working storage
DFHEISTG DSECT
TRANSARG DS    XL(AXLLEN)          Argument for Translate
CPTSBUFF DS    XL80                Data buffer for Message
CPTLENG  DS    H                   LENGTH OF INPUT MESSAGE
*
label    DFHEIENT
           ...
          L    R9,ACMTOKEN         Load ACM Token
           ...
*
*        CPT Translate
*
TRANS    DS     0H
         USING  AXL,TRANSARG
*
         MVI   AXLLTYP1,AXLTATOE   ASCII TO EBCDIC FLAG
         MVI   AXLXMOD1,AXLMSBCS   CHARACTER MODE (SBCS)
         LA    R04,CPTSBUFF        LOAD SEND BUFFER ADDRESS
         ST    R04,AXLSADDR        SAVE SOURCE ADDRESS
         LH    R05,CPTLENG         LOAD LENGTH OF SOURCE
         ST    R05,AXLSLENG        SAVE SOURCE LENGTH
         ST    R9,AXLTOKEN         Save connection token
*
         LA    R01,TRANSARG        Point to control block
         ST    R01,CPTPARMS        Store address of control block
         T09MCALL TRANS,PARM=CPTPARMS
         LTR   R15,R15             Test Return Code
         BZ    TRANSOK             Good,
*                                  Bad... Translate error
```

```
          L      R3,AXLRTNCD         Load AXL Return Code
          L      R4,AXLDGNCD         Load AXL Diagnostic Code
```

Outbound Translation Example

In this example, the application needs to send data to a remote ASCII host. It translates data from EBCDIC to ASCII by setting the AXLTETOA flag. The token AXLTOKEN, data buffer address AXLSADDR, and length to translate, AXLSLENG are set in the AXL. The default translation mode of SBCS is selected.

The application checks the AXLRTNCD return code to determine the TRANSLATE service completion status.

Note: Relevant parameters in the example are in **bold**:

```
*        Dsect's

         T09DACM MF=DSECT          Argument for Connection Management
         T09DAXL MF=DSECT          Argument for Translate
*
*        Working storage
DFHEISTG DSECT
TRANSARG DS    XL(AXLLEN)          Argument for Translate
*
CPTSBUFF DS    XL80                Data buffer for Message
CPTLENG  DS    H                   LENGTH OF INPUT MESSAGE
*
label    DFHEIENT
           .
           . CPT Connection Management initialization and request
           .
           L    R9,ACMTOKEN        Load ACM Token
           .
           .
           .
*
*       CPT Translate
*
TRANS    DS     0H
         USING  AXL,TRANSARG
*
         MVI   AXLLTYP1,AXLTETOA   EBCDIC TO ASCII FLAG
         MVI   AXLXMOD1,AXLMSBCS   CHARACTER MODE (SBCS)
         LA    R04,CPTSBUFF        LOAD SEND BUFFER ADDRESS
         ST    R04,AXLSADDR        SAVE SOURCE ADDRESS
         LH    R05,CPTLENG         LOAD LENGTH OF SOURCE
         ST    R05,AXLSLENG        SAVE SOURCE LENGTH
         ST    R9,AXLTOKEN         Save connection token
*
         LA    R01,TRANSARG        Point to control block
         ST    R01,CPTPARMS        Store address of control block
         T09MCALL TRANS,PARM=CPTPARMS
         LTR   R15,R15             Test Return Code
         BZ    CALLSEND            Good,
*
*         Translate error
*
         L      R3,AXLRTNCD        Load AXL Return Code
         L      R4,AXLDGNCD        Load AXL Diagnostic Code
```

Custom Translation Table Usage Notes

Unicenter SOLVE:CPT fully supports customizing translation tables to fit your environment. For information and customization instructions, see the "Translation Tables" chapter in the *Administrator Guide*.

Once you have customized a translation table for your environment, you still need to tell Unicenter SOLVE:CPT that you want to use it.

Set the Default Translation Table

Configure the most heavily used the translation table to be the default Unicenter SOLVE:CPT's translation table.

To do this, your Unicenter SOLVE:CPT administrator needs to change the TRANSTBL parameter on the T09MCICS macro in the configuration table T09CON*xx* to the name of your customized translation table.

For information on changing the default name, see the T09MCICS macro section of the "Configuration Reference" chapter in the *Administrator Guide*.

The new translation table must be in CICS' DFHRPL concatenation in order to be loaded by Unicenter SOLVE:CPT.

Overriding the Default Translation Table

This method uses a translation table other than your Unicenter SOLVE:CPT system-wide default.

To override the default translation table:

1. See the "Translation Tables" chapter of the *Administrator Guide*. Customize a translation table.
2. Move the table from Step 1 into a library in the CICS DFHRPL concatenation.
3. Issue the following command in your program to load the table:

   ```
   EXEC  CICS LOAD
         PROGRAM(mytable)
         SET(ptr-ref)
         NOHANDLE
   ```

4. Copy the address ptr-ref to the AXLTABLE parameter.
5. Call the Unicenter SOLVE:CPT TRANSLATE service.
6. Issue the following command in your program to release the storage for the table.

   ```
   EXEC  CICS RELEASE
        PROGRAM(mytable)
         NOHANDLE
   ```

For performance or efficiency reasons, if you use the table for multiple transactions, you should consider using the HOLD keyword on the EXEC CICS LOAD command. You will need to store the address of the table for others to use. You have to manage the EXEC CICS RELEASE carefully so that storage cleanup is eventually done.

Parameter Values Returned in the AXL

After the TRANSLATE call returns control to your application program, the following fields are propagated with valid completion information. These updated values are passed back to the application in the AXL control block.

Field Name	Description
AFTDGNCD	Diagnostic code.
AFTRTNCD	Return code.

Assembler DSECTs

Sample Assembler DSECTs are provided in the distributed software and are available to you in ***cpthlq***.T09MAC. Variable field names contained in the distributed samples and the examples in this guide refer to these DSECTs.

T09DAXL — Assembler DSECT name for the AXL. For detailed information and a sample copy of the Assembler DSECT, see the AXL: Argument for TRANSLATE Used by the TRANSLATE Service section in the "Control Block Layouts" appendix.

All Assembler constants that apply to AXL calls are imbedded in the AXL DSECT sample.

Sample Programs

Sample Assembler source code is available in the distributed software in the ***cpthlq***.T09SAMP library. You should be able to find a sample that matches your programming requirement. For complete details on the function a sample program provides, see the program descriptions in the "Unicenter SOLVE:CPT API Services" chapter and the descriptions of the sample members listed below.

Name	Description
T09PACLU	UDP client program
T09PACL1	Client Application sends typed in data to the server waiting for the information to be echoed back from the server.
T09PACL2	Client Application to send an internal message using either the FULL, SEP or LL to be echoed back by the server.
T09PASC1	Sample Security exit

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, an error associated with translation occurred. The general return code (AXLRTNCD) in register 15 and the diagnostic code (AXLDGNCD) in register zero indicate the nature of the failure.

Return Codes

The TRANSLATE service returns a code in registers R15 and R0 indicating the results of the execution. These values are in the AXLRTNCD (R15) and AXLDGNCD (R0).

DSECT T09DRTCD contains equates and descriptions for the possible return codes. T09DRTCD is available in the distributed software in ***cpthlq***.T09MAC. See the appendix "Return Codes" for a sample copy of the T09DRTCD DSECT.

The following table lists the return codes that can apply to the TRANSLATE call.

Decimal	Hex	Diagnostic Code	Variable	Description
0	0	No	CPTIRCOK	Request completed successfully.
17	11	No	CPTEVRSN	Control block version number not supported.
20	14	No	CPTETOKN	Specified data transfer token is invalid.
21	15	No	CPTEBUFF	Buffer address or length invalid.
22	16	No	CPTECHAR	Translate character set is invalid.
23	17	No	CPTEMODE	Translate mode specification is invalid.
25	19	No	CPTETABL	Specified table is not correct.
31	1F	No	CPTEFRMT	Other Socket Call Parameter List format or specification error.
34	22	No	CPTENAPI	API not fully available; retry.
40	28	Yes	CPTETERM	Environment is being terminated.
254	FE	Is Abend Code	CPTABEND	Abnormal termination. **Note:** The diagnostic code is the abnormal termination code that is normally a CICS abend code, but can also be in the "Abend Codes" chapter of the *Message Guide*.
255	FF	No	CPTEOTHR	Other error.

Usage Notes

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

The version number, AXLVERS, indicates the Unicenter SOLVE: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 processing the request.

The function code, AXLFUNC, indicates the Unicenter SOLVE: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. It sets single, double or mixed character set translation. Currently, only single-byte character set translation, which is the default, is supported.

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.

Complete Parameter List

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: 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.
AXLLTYPE	Specifies TRANSLATE service translation type or direction. AXLTATOE — Indicates ASCII to EBCDIC translation. AXLTAUPC — Indicates ASCII to uppercase ASCII translation. AXLTETOA — Indicates EBCDIC to ASCII translation. AXLTEUPC — Indicates EBCDIC to uppercase EBCDIC translation. Default: None.
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: None.
AXLSADDR	Required. Source text buffer address. Indicates the address of the user data to translate. It 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: None.
AXLSLENG	Required. Source text buffer length. Indicates the length, in bytes of user data in the storage area, as identified by the AXLSADDR field. **Note:** A zero value causes the request to fail. Default: None.

AXLTABLE	Address of user translation table. See Custom Translation Table Usage Notes for notes on usage of a customized table. Default: None.
AXLTOKEN	Connection token. Specifies the token, which represents either a TCP or UDP connection or UDP. Default: None.
AXLVERS	Required. Version number indicates the Unicenter SOLVE:CPT version number of the argument used by the calling program. **Note:** Must be set to a binary two for this release of Unicenter SOLVE:CPT. Default: None.
AXLXMODE	Specifies TRANSLATE service translation mode or character set.

AXLMDBCS	Indicates double-byte character set translation. **Note:** This option is currently not supported.
AXLMMIXD	Indicates mixed mode character set translation. This mode specifies single- and double-byte translation. **Note:** This option is currently not supported.
AXLMNUMS	Indicates numeric set translation. **Note:** This option is currently not supported.
AXLMSBCS	Indicates single-byte character set translation.

Default: AXLMSBCS.

Appendix

A Return Codes

This appendix provides you with information about the Unicenter SOLVE:CPT return codes.

The Return Code table below provides you with:

- A description of the meaning of the return code
- The decimal and hexadecimal values of the return code
- A variable field name from the Assembler DSECT

Immediately following the RC table is the T09DRTCD Assembler DSECT. It contains convenient variable names for you to use in your application program. This DSECT is used by all the provided application programming samples. The T09DRTCD DSECT is located in the ***cpthlq.***T09MAC distributed library.

Return Codes

Decimal	Hex	Diagnostic Code	Variable	Description
0	0	No	CPTIRCOK	Request completed successfully.
1	1	No	CPTWTIMEO	Timed receive call timed out.
4	4	No	CPTWNEGO	System limits applied to buffer or Queue sizes.
6	6	Yes	CPTWBLCK	A non-blocking call to a service detected a wait condition.
10	A	No	CPTWNSEP	Separator type receive found no separator characters.
15	0F	Yes	CPTWEXCP	Other warning.
17	11	No	CPTEVRSN	Control block version number not supported.

Decimal	Hex	Diagnostic Code	Variable	Description
18	12	Yes	CPTECONN	Required Parameter not passed. E.g. host, port, …
19	13	No	CPTEPROT	Specified protocol not supported.
20	14	No	CPTETOKN	Specified data transfer token is invalid.
21	15	No	CPTEBUFF	Buffer address and/or length invalid.
22	16	No	CPTECHAR	Translate character set is invalid.
23	17	No	CPTEMODE	Translate mode specification is invalid.
24	18	Yes	CPTECOPT	Close mode specification is invalid.
25	19	Yes	CPTETABL	Specified translate table not correct.
26	1A	Yes	CPTETRID	Designated transaction ID cannot start.
27	1B	No	CPTETIME	Receive timeout value not specified.
28	1C	No	CPTESEP#	Receive type separator number of separator characters equal to 1 or 2.
29	1D	No	CPTEOPTN	Receive options selected is a combination which is invalid.
30	1E	No	CPTEOPRL	Receive option not supported by transport carrier.
31	1F	No	CPTEFRMT	Other Socket Call Parameter List format or specification error.
33	21	Yes	CPTEPBSY	Selected port is busy with active server.
34	22	No	CPTENAPI	API not fully available; retry.
35	23	Yes	CPTENAVL	Requested facility is not available.
36	24	Yes	CPTEDRAN	TCP/IP environment is terminating.

Decimal	Hex	Diagnostic Code	Variable	Description
37	25	No	CPTESLCT	Select tool transaction is not running.
38	26	No	CPTERCVT	Receive tool not defined in the T09CON*xx* CPT configuration table.
40	28	Yes	CPTETERM	Environment is being terminated.
46	2E	No	CPTESCTY	Security Exit terminated the session.
47	2F	Yes	CPTEENVR	Other transport layer environmental condition.
65	41	Yes	CPTERLSE	Orderly release of remote connection request.
68	44	Yes	CPTEDISC	Remote connection not available or aborted.
72	48	Yes	CPTEPRGE	Remote connection environment terminating.
79	4F	Yes	CPTEINTG	Other transport layer connection/data integrity error.
138	8A	No	CPTEWECB	ECB already being waited on
143	8F	Yes	CPTEPROC	Procedural error.
254	FE	Is abend code	CPTABEND	Abnormal termination. The diagnostic code is the abnormal termination code, which is normally a CICS abend code, but can also be in the "Abend Codes" chapter of the *Message Guide*.
255	FF	No	CPTEOTHR	Other error.

Diagnostic Code Field

The diagnostic code field depends on the error event recorded in the Unicenter SOLVE:CPT return code field. The diagnostic code field could be CICS abend code, ERRNO, or other value depending on the Unicenter SOLVE:CPT return code failure.

When a Unicenter SOLVE:CPT API call fails, the product prefers to return the return code and diagnostic code field pair from the first error event that occurred during the Unicenter SOLVE:CPT API call.

An API system Error return code (ERRNO) can be mapped back into a Unicenter SOLVE:CPT return code when an EZASOKET or EZACICAL error occurs during processing of a Unicenter SOLVE:CPT API call. If the first error on a Unicenter SOLVE:CPT API call is an EZASOKET or EZACICAL error then the return code contains the TCP API system Error return code (ERRNO). To determine the meaning of the ERRNO number, See IBM's *Communication Server IP API Guide* or IBM's *Communication Server IP CICS Sockets Guide* or equivalent.

Assembler MACRO T09DRTCD

```
         MACRO
         T09DRTCD ,
***********************************************************************
*        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'0A'    SEP TYPE RECV FOUND NO SEPERATOR CHARACTERS
CPTWEXCP EQU   X'0F'    TPL EXCEPTIONAL CONDITION
***********************************************************************
*        CONTROL BLOCK ARGUMENT ERRORS                                *
***********************************************************************
CPTEVRSN 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 CHARS = 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
CPTENAPI EQU   X'22'    SNS/API/CICS NOT FULLY AVAILABLE, RETRY
CPTENAVL EQU   X'23'    REQUESTED FACILITY IS NOT AVAILABLE
CPTEDRAN EQU   X'24'    ENVIRONMENT IS BEING DRAINED
CPTESLCT EQU   X'25'    SELECT TOOL TRANSACTION NOT RUNNING
CPTERCVT EQU   X'26'    RECEIVE TOOL NOT DEFINED
CPTETERM EQU   X'28'    ENVIRONMENT IS BEING TERMINATED
CPTESCTY EQU   X'2E'    SECURITY EXIT CANCELLED THE SESSION
CPTEENVR EQU   X'2F'    OTHER TPL ENVIRONMENTAL CONDITION
***********************************************************************
*        CONNECTION ERRORS                                            *
***********************************************************************
CPTERLSE EQU   X'41'    ORDERLY RELEASE OF REMOTE CONNECTION REQUESTED
CPTEDISC EQU   X'44'    REMOTE CONNECTION 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                         *
***********************************************************************
CPTEWECB EQU   X'8A'    ECB ALREADY BEING WAITED ON
CPTEPROC EQU   X'8F'
***********************************************************************
*        ABNORMAL ENVIRONMENTAL ERROR FORCING CI00 ABEND              *
***********************************************************************
CPTABEND EQU   X'FE'
***********************************************************************
*        ANY OTHER CURRENTLY UNDEFINED CONDITION                      *
***********************************************************************
CPTEOTHR EQU   X'FF'
         MEND
```

Appendix

B Control Block Layouts

This appendix describes these Unicenter SOLVE:CPT control blocks.

The following topics are discussed in this appendix:

- ACL: Argument for CLose Used by the CLOSE API Service
- ACM: Argument for Connection Management Used by the CONNECT and LISTEN API Services
- ADT: Argument for Data Transfer Used by the RECEIVE, SEND, RECVFROM, and SENDTO Services
- AFM: Argument for Facility Management Used by the GIVE and TAKE Services
- AFT: Argument for File Transfer Used by the FTP Client Service Call
- AXL: Argument for Data Translation Used by the Translate API Service
- Client Data Listener Transaction Start
- Connection Time Security Program Control Block
- Parameter List Passed to T09MTRAN Initiated Transactions
- LCA0000 and CFG0000 Control Blocks

ACL: Argument for CLose Used by the CLOSE API Service

This section describes the Unicenter SOLVE:CPT Argument for CLose, the ACL. The ACL is used by the CLOSE service to terminate TCP connections.

It provides the following information:

- An offset table of the ACL fields
- An alphabetical list of ACL fields
- A sample copy of the T09DACL DSECT

Note: The ACL control block is 28 bytes in length, which is x'1C' in hexadecimal. The space for this control block must be created by the application and mapped to by the sample T09DACL DSECT.

Offset Table

This table provides information from the T09DACL DSECT member with field descriptions.

Decimal	Hex	Type	Length	Name	Description
0	(0)			ACL	Transport endpoint exit parameters.
0	(0)	HALF WORD	4	ACLVERS	Version number.
2	(2)	HALF WORD	4	ACLFUNC	Function code.
4	(4)	ADDRESS	4	ACLTOKEN	Token (CEP).
8	(8)	ADDRESS	4		Reserved.
12	(C)	FULL WORD	4		Reserved.
16	(10)	FULL WORD	4	ACLRTNCD	Return code.
20	(14)	FULL WORD	4	ACLDGNCD	Diagnostic code.
24	(18)	FULL WORD	4	ACLOPCDS	Termination Option Codes.

Decimal	Hex	Type	Length	Name	Description
24	(18)	BYTE	1	ACLOPCD4	Termination Option Code 4.
25	(19)	BYTE	1	ACLOPCD3	Termination Option Code 3.
26	(1A)	BYTE	1	ACLOPCD2	Termination Option Code 2.
27	(1B)	BYTE	1	ACLOPCD1	Termination Option Code 1.
		....		ACLORDER	- Orderly release.
		1		ACLABORT	- Abortive release.
		...1		ACLFCLSE	- Abortive/fast termination.
		..1.		ACLFFREE	- DEQ/FREEMAIN termination.
28	(1C)			ACLLEN	Length of ACL.

Alphabetized Field Name Cross-Reference Table

This table is an alphabetized list of field names mapped to the offset within the ACL control block.

Name	Hex Offset	Hex Value
ACLABORT	1B	01
ACLDGNCD	14	
ACLFCLSE	1B	10
ACLFFREE	1B	20
ACLFUNC	2	
ACLLEN	1C	1C
ACLOPCDS	18	
ACLOPCD1	1B	
ACLOPCD2	1A	
ACLOPCD3	19	
ACLOPCD4	18	

Name	Hex Offset	Hex Value
ACLORDER	1B	00
ACLTOKEN	4	
ACLVERS	0	

Sample DSECT Member T09DACL

The following is a sample of DSECT member T09DACL located in the ***cpthlq.***T09MAC distributed library.

- It contains the layout and field names for your use in your application program
- It is used by all the provided application programming samples

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

```
Name     Operation Operands Description

ACL      DSECT              ARGUMENT FOR CONNECTION RELEASE
ACLVERS  DS        H        VERSION NUMBER
ACLFUNC  DS        H        FUNCTION CODE
ACLTOKEN DS        A        TOKEN (CEP)
ACLINTCB DS        A        INTERNAL CB, CA use only
         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
ACLC2INC EQU       X'80'    - INTERNAL CLOSE CALL
ACLOPCD1 DS        X        TERMINATION OPTION CODE 1
ACLORDER EQU       X’00’    - ORDERLY RELEASE
ACLABORT EQU       X’01’    - ABORTIVE RELEASE
ACLSHUT0 EQU       X’02’    - SHUTDOWN DISALLOW RECVS ON SOCKET
ACLSHUT1 EQU       X’04’    - SHUTDOWN DISALLOW SENDS ON SOCKET
ACLSHUT2 EQU       X’08’    - SHUTDOWN DISABLE SENDS AND RECVS
ACLFPURG EQU       X’40’    - CLOSE EP, BUT DON’T FREE
ACLFFREE EQU       X’20’    - ABORTIVE/FAST TERMINATION
ACLFCLSE EQU       X’10’    - DEQ/FREEMAIN TERMINATION
ACLTIMEO DS        H        TIME TO LINGER ON CLOSE
*
ACLLEN   EQU       *-ACL    LENGTH OF ACL
```

ACM: Argument for Connection Management Used by the CONNECT and LISTEN API Services

This section describes the Unicenter SOLVE:CPT Argument for Connection Management, the ACM. The ACM is used by the CONNECT and LISTEN services to establish TCP connections.

It provides the following information:

- An offset table of the ACM fields
- An alphabetical list of ACM fields
- A sample copy of the T09DACM DSECT

Note: The ACM control block is 628 bytes in length, which is x'274' in hexadecimal. The space for this control block must be created by the application and mapped to by the sample T09DACM DSECT.

Offsets

This table provides information from the T09DACM DSECT member with field descriptions.

Decimal	Hex	Type	Length	Name	Description
0	(0)			ACM	Transport endpoint exit parameters.
0	(0)	HALF WORD	2	ACMVERS	Version number.
2	(2)	HALF WORD	2	ACMFUNC	Function code.
4	(4)	ADDRESS	4	ACMTOKEN	Token (CEP).
8	(8)	ADDRESS	4		(Reserved).
12	(C)	FULL WORD	4		(Reserved).
16	(10)	FULL WORD	4	ACMRTNCD	Return code.
20	(14)	FULL WORD	4	ACMDGNCD	Diagnostic code.
24	(18)	FULL WORD	4	ACMSTATS	Statistic flags.
24	(18)	BYTE	1	ACMSTAT4	Statistic flag 4.
25	(19)	BYTE	1	ACMSTAT3	Statistic flag 3.
26	(1A)	BYTE	1	ACMSTAT2	Statistic flag 2.

Decimal	Hex	Type	Length	Name	Description
27	(1B)	BYTE	1	ACMSTAT1	Statistic flag 1.
		1		ACMSCONN	- Connection statistics.
		.... ..1.		ACMSTERM	- Termination statistics.
28	(1C)	FULL WORD	4	ACMTRACE	Trace flags.
32	(20)	FULL WORD	4	ACMQSEND	TSEND queue size.
36	(24)	FULL WORD	4	ACMMSEND	Maximum TSEND TPL buffer size.
40	(28)	FULL WORD	4	ACMQRECV	TRECV queue size.
44	(2C)	FULL WORD	4	ACMMRECV	Maximum TRECV .TPL buffer size.
48	(30)	FULL WORD	4	ACMTLSTN	Listen Token.
52	(34)	FULL WORD	4	ACMUCNTX	User context field.
56	(38)	CHARACTER	4	ACMTRNID	Transaction ID.
60	(3C)	BYTE	1		Reserved for C String.
61	(3D)	BYTE	3		Unused.
64	(40)	HALF WORD	2	ACMLPORT	Local Port.
66	(42)	HALF WORD	2	ACMRPORT	Remote Port.
68	(44)	CHARACTER	36	ACMSRVCE	Transport Service .Name.
104	(68)	BYTE	1		Reserved for C String.
105	(69)	BYTE	3		Unused.
108	(6C)	ADDRESS	4	ACMLADDR	Local IP Address.
112	(70)	ADDRESS	4	ACMRADDR	Remote IP Address.
116	(74)	CHARACTER	255	ACMLNAME	Local IP Host Name.
371	(173)	BYTE	1		Reserved for C String.
372	(174)	CHARACTER	255	ACMRNAME	Remote IP Host Name.

Decimal	Hex	Type	Length	Name	Description
627	(273)	BYTE	1		Reserved for 'C' String.
628	(274)			ACMLEN	Length of ACM.

Alphabetized Field Name Cross-Reference Table

This table provides an alphabetized list of field names mapped to the offset within the ACM control block.

Name	Hex Offset	Hex Value
ACMDGNCD	14	
ACMFUNC	2	
ACMLADDR	6C	
ACMLEN	74	
ACMLNAME	74	
ACMLPORT	40	
ACMMRECV	2C	
ACMMSEND	24	
ACMQRECV	28	
ACMQSEND	2	
ACMRADDR	70	
ACMRNAME	174	
ACMRPORT	42	
ACMRTNCD	10	
ACMSCONN	1B	01
ACMSRVCE	44	
ACMSTAT1	1B	
ACMSTAT2	1A	
ACMSTAT3	19	
ACMSTAT4	18	
ACMSTATS	18	

Name	Hex Offset	Hex Value
ACMSTERM	1B	02
ACMTOKEN	4	
ACMTRAC1	1F	
ACMTRAC2	1E	
ACMTRAC3	1D	
ACMTRAC4	1C	
ACMTRACE	1C	
ACMTRNID	38	
ACMUCNTX	34	
ACMVERS	0	

Sample DSECT Member T09DACM

The following is a sample of DSECT member T09DACM located in the ***cpthlq.***T09MAC distributed library.

- It contains the layout and field names for your use in your application program
- It is used by all the provided application programming samples

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

```
Name     Operation Operands Description

ACM      DSECT              ARGUMENT FOR CONNECTION MANAGEMENT
*
ACMVERS  DS        H        VERSION NUMBER
ACMVERSN EQU       2        THIS IS VERSION NUMBER 2
ACMFUNC  DS        H        FUNCTION CODE
ACMTOKEN DS        A        TOKEN (CEP)
ACMCDTBL DS        CL8      OPT. TRANSLATE TBL FOR CLIENT
ACMRTNCD DS        F        RETURN CODE
ACMDGNCD DS        F        TCP API errno value
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
*
*        The ACMTRAC2 & 1 flags no longer effect CPT behavior.
*        They have been left in to prevent compile errors.
*
ACMTRAC2 DS        X        SECOND TRACE BYTE
ACMTRAC1 DS        X        FIRST TRACE BYTE
```

```
*
ACMQSEND DS         F         TSEND QUEUE SIZE
ACMMSEND DS         F         MAX TSEND TPL BUFFER SIZE
ACMQRECV DS         F         TRECV QUEUE SIZE
ACMMRECV DS         F         MAX 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 SERVCIE NAME
*
*        The ACMSRVCE control block is no longer supported.
*        It will be reused to support other parameters.
*
         ORG        ACMSRVCE
         DS         XL32
*
*       The ACMINTCB and ACMO3INC fields are reserved for
*       internal CPT usage.
*
ACMINTCB DS         A         INTERNAL CB, CA use only
         ORG ,
         DS         X         RESERVED FOR C STRING
ACMOPTN3 DS         X         ACM OPTION FLAG 3
ACM03INC EQU        X'80'     - Internal Call CA use only
ACMOPTNS DS         0H        ACM OPTION FLAGS
ACMOPTN2 DS         X         ACM OPTION FLAG 2
ACMEZALS EQU        X'80'     - CICS SOCKETS COMPATABILITY
ACMSLCL  EQU        X'20'     - USE LOC PORT IN ACMLPORT
ACMPARM  EQU        X'10'     - RECEIVE PARM PRESENT
ACMOMRO  EQU        X’08’     - LISTEN DONE VIA CPT/MRO
ACMOSEC  EQU        X’04’     - SECURITY NAME PRESENT
ACMOUSR  EQU        X’02’     - USERID FIELD PRESENT
ACMOCLEN EQU        X’01’     - CLIENT DATA LENGTH PRESENT
ACMOPTN1 DS         X         ACM OPTION FLAG 1
ACMCTRAN EQU        X’80’     - TRANSLATE CLIENT/DATA
ACMOTRAN EQU        X’40’     - OPT. TRANS. TBL. FOR C/D
ACMNBLKO EQU        X’08’     - NON BLOCKING LISTEN/CONNECT IUCV
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 FR 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
         ORG        ACMLNAME
ACMSECLM DS         CL8       SECURITY PROGRAM
ACMUSRID DS         CL8       USERID TO START NEXT TRAN
ACMCLNTL DS         AL4       CLIENT DATA LENGTH
ACMMROEP DS         F         ENDPOINT ID FROM CPT/MRO
ACMMROAS DS         A         ASCB ADDRESS FROM CPT/MRO
ACMCLNT  DS         CL40      CLIENT STRUCTURE GETCLIENTID
ACMIDENT DS         CL16      TAKESOCKET IDENT   NAME
ACMTSUBT DS         CL8       TAKESOCKET SUBTASK NAME
ACMTEPID DS         H         Takesocket EPID
ACMIMSOC DS         H         TAKESOCKET MAX NUM SOCKET PASS
         ORG ,
ACMRNAME DS         CL255     REMOTE IP HOST NAME
         DS         X         RESERVED FOR C STRING
```

```
ACMMSOCK DS        H        MAX SOCKETS TO ALLOC.
ACMBCKLG DS        H        Q LISTEN BACKLOG QUEUE SIZE
ACMTIMEO DS        F        TIMEOUT TO RECV CLIENT DATA
         DS        F
*
ACMLEN   EQU       *-ACM    LENGTH OF ACM
```

ADT: Argument for Data Transfer Used by the RECEIVE, SEND, RECVFROM, and SENDTO Services

This section describes the Unicenter SOLVE:CPT ADT. The ADT is used by the RECEIVE, SEND, RECVFROM, and SENDTO services to transfer data and provides the following information:

- An alphabetical list of ADT fields
- A sample copy of the T09DADT DSECT
- An offset table of the ADT fields

Note: The ADT control block is 644 bytes in length, which is x'284' in hexadecimal. The space for this control block must be created by the application and mapped to by the sample T09DADT DSECT.

Offsets

This table provides information from the T09DADT DSECT member with field descriptions.

Decimal	Hex	Type	Length	Name	Description
0	(0)			ADT	Data transfer parameters.
0	(0)	HALF WORD	2	ADTVERS	Version number.
2	(2)	HALF WORD	2	ADTFUNC	Function code.
4	(4)	ADDRESS	4	ADTTOKEN	Token (CEP).
8	(8)	ADDRESS	4	ADTBUFFA	Data buffer address.
12	(C)	FULL WORD	4	ADTBUFFL	Data buffer length.
16	(10)	FULL WORD	4	ADTRTNCD	Return code.
20	(14)	FULL WORD	4	ADTDGNCD	Diagnostic code.
24	(18)	FULL WORD	4	ADTOPTNS	Options (reserved).
24	(18)	BYTE	1	ADTOPCD4	Option 4.

Decimal	Hex	Type	Length	Name	Description
25	(19)	BYTE	1	ADTOPCD3	Option 3.
26	(1A)	BYTE	1	ADTOPCD2	Option 2.
27	(1B)	BYTE	1	ADTOPCD1	Option 1.
		1		ADTFVLST	Vector list flag.
28	(1C)			ADTLEN	Length of ADT.

Alphabetized Cross-Reference Table

This table provides an alphabetized list of field names mapped to the offset within the ADT control block.

Name	Hex Offset	Hex Value
ADTBUFFA	8	
ADTBUFFL	C	
ADTDGNCD	14	
ADTFUNC	2	
ADTFVLST	1B	01
ADTLEN	1C	
ADTOPCD4	1B	
ADTOPCD3	1A	
ADTOPCD2	19	
ADTOPCD1	18	
ADTOPTNS	18	
ADTRTNCD	10	
ADTTOKEN	4	
ADTVERS	0	

Sample DSECT Member T09DADT

The following is a sample of DSECT member T09DACM located in the ***cpthlq.***T09MAC distributed library.

It provides the following information:

- It contains the layout and field names for your use in your application program
- It is used by all the provided application programming samples

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

```
Name     Operation Operands Description

ADT      DSECT              ARGUMENT FOR CONNECTION MANAGEMENT
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
ADTSCONN EQU       X'01'    - CONNECTION STATISTICS
ADTSTERM EQU       X'02'    - TERMINATION STATISTICS
*
ADTTRACE DS        0F       TRACE FLAG
         DS        XL2
*
* Note the following trace flags have been replaced
*      with TCPeep tracing - see Administrator Guide
*
ADTTRAC2 DS        X        SECOND TRACE BYTE (FOR HI LVL LANG)
ADTTRAC1 DS        X        FIRST TRACE BYTE (FOR HI LVL LANG)
*
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        WAIT SECONDS FOR TIMED RCV/SELECT
         DS        F        MILLESECONDS
ADTVECTR DS        A        ADDRESS OF SELECT VECTOR
ADTSLCTR EQU       X'01'    SELECT ON READY FOR READ
ADTSLCTW EQU       X'02'    SELECT ON READY FOR WRITE
ADTSLCTE EQU       X'04'    SELECT ON EXCEPTIONAL CONDITION
ADTNSLCT DS        F        NUMBER OF ENTRIES IN THE VECTOR
ADTSLCTD DS        F        NUMBER OF SOCKETS SELECTED
ADTLPORT DS        H        LOCAL TRANSPORT PROVIDER PORT
ADTRPORT DS        H        REMOTE TRANSPORT PROVIDER PORT
ADTMSOCK DS        H        MAX SOCKETS TO ALLOC. FOR IUCV PATH
*
```

```
* Note the following is no longer supported
*
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
ADTNWAIT EQU       X’20’    DO NOT WAIT - SELECT TOOL
ADTNOQUE EQU       X'10'    DO NOT QUEUE API RECEIVES
ADTRT100 EQU       X'08'    RECV TIMEOUT in 1/100 second
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)
ADTNBLKS EQU       X’20’    DO NOT BLOCK ON SEND/SENDTO (IBM)
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
*
ADTLEN   EQU       *-ADT    LENGTH OF ADT
```

AFM: Argument for Facility Management Used by the GIVE and TAKE Services

This section describes the Unicenter SOLVE:CPT AFM. The AFM is used by the GIVE and TAKE calls to transfer ownership of a TCP connection between two CICS transactions. The following information is provided:

- An offset table of the AFM fields
- An alphabetical list of AFM fields
- A sample copy of the T09DAFM DSECT

Note: The AFM control block is 40 bytes in length, which is x'28' in hexadecimal. The space for this control block must be created by the application and mapped to by the sample T09DAFM DSECT.

Offsets

This table provides information from the T09DAFM DSECT member with field descriptions.

Decimal	Hex	Type	Length	Name	Description
0	(0)			AFM	Data transfer parameters.
0	(0)	HALF WORD	2	AFMVERS	Version number.
2	(2)	HALF WORD	2	AFMFUNC	Function code.
4	(4)	ADDRESS	4	AFMTOKEN	Token (CEP).
8	(8)	ADDRESS	4		Reserved.
12	(C)	FULL WORD	4		Reserved.
16	(10)	FULL WORD	4	AFMRTNCD	Return code.
20	(14)	FULL WORD	4	AFMDGNCD	Diagnostic code.
24	(18)	FULL WORD	4	AFMOPTNS	Options (reserved).
24	(18)	BYTE	1	AFMOPCD4	Option 4.
25	(19)	BYTE	1	AFMOPCD3	Option 3.
26	(1A)	BYTE	1	AFMOPCD2	Option 2.
27	(1B)	BYTE	1	AFMOPCD1	Option 1.
28	(1C)			AFMLEN	Length of AFM.

Alphabetized Cross-Reference Table

This table provides an alphabetized list of field names mapped to the offset within the AFM control block.

Name	Hex Offset	Hex Value
AFMDGNCD	14	
AFMFUNC	2	
AFMLEN	1C	
AFMOPCD1	1B	
AFMOPCD2	1A	
AFMOPCD3	19	
AFMOPCD4	18	
AFMOPTNS	18	
AFMRTNCD	10	
AFMTOKEN	4	
AFMVERS	0	

Sample DSECT Member T09DAFM

The following is a sample of DSECT member T09DAFM located in the ***cpthlq.***T09MAC distributed library. It:

- Contains the layout and field names for your use in your application program
- Is used by all the provided application programming samples

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

```
Name     Operation Operands Description

AFM      DSECT              ARGUEMENT FACILITY MANAGEMENT
AFMVERS  DS        H        VERSION NUMBER
AFMFUNC  DS        H        FUNCTION CODE
AFMTOKEN DS        A        TOKEN (CEP)
AFMNTRAN DS        CL4      NEXT TRANS ID TO START
         DS        F        RESERVED
AFMRTNCD DS        F        RETURN CODE
AFMDGNCD DS        F        DIAGNOSTIC CODE
AFMOPTNS DS        0F       FACILITY MGMT 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
AFMOPDEQ EQU       X’80’    - DEQUEUE TOKEN ONLY
AFMOPENQ EQU       X’40’    - ENQUEUE TOKEN ONLY
AFMOPSEL EQU       X’20’    - PASS TOKEN TO SELECT TOOL
AFMOPCOM EQU       X’10’    - COMMAREA / LENGTH PASSED
AFMMSOCK DS        H        MAX SOCKETS ON TAKE
AFMCOMMA DS        A        COMMAREA ADDRESS
AFMCOMML DS        F        COMMAREA LENGTH
AFMLEN   EQU       *-AFM    LENGTH OF AFM
```

AFT: Argument for File Transfer Used by the FTP Client Service Call

This section describes the Unicenter SOLVE:CPT Argument for File Transfer, the AFT. The AFT is used by the FTP client service call to define the arguments used to make a FTP client call to a remote FTP server. The following information is provided:

- An offset table of the AFT fields
- An alphabetical list of AFT fields
- A sample copy of the T09DAFT DSECT

Note: The AFT control block is 320 bytes in length, which is x'140' in hexadecimal. The space for this control block must be created by the application and mapped to by the sample T09DAFT DSECT.

Offsets

This table provides information from the T09DAFT DSECT member with field descriptions.

Decimal	Hex	Type	Length	Name	Description
0	(0)			AFT	File Transfer parameters

Alphabetized Cross-Reference Table

This table provides an alphabetized list of field names mapped to the offset within the AXL control block.

Name	Hex Offset	Hex Value
AFTDGNCD	14	

Sample DSECT Member T09DAFT

The following is a sample of DSECT member T09DAFT located in the ***cpthlq.***T09MAC distributed library.

It:

- Contains the layout and field names for your use in your application program
- Is used by all the provided application programming samples

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

```
Name     Operation Operands Description

CFCAFT   DSECT              ARGUMENT FOR FILE TRANSFER
AFTVERS  DS        H        VERSION NUMBER
AFTVERSN EQU       2        THIS IS VERSION NUMBER 2
AFTOPTNS DS        0XL2     FTP ACM OPTION FLAGS
AFTOPTN2 DS        X        FTP OPTION FLAG 2
AFTOPTN1 DS        X        FTP OPTION FLAG 1
AFTNODNR EQU       4        TURN OFF DNR
AFTRNAMA DS        A        ADDR OF REMOTE IP HOST NAME
AFTRNAML DS        F        LEN OF REMOTE IP HOST NAME
AFTRADDR DS        A        REMOTE IP HOST ADDRESS
AFTUSER  DS        CL64     USERID FOR REMOTE LOGON
AFTPASS  DS        CL64     PASSWORD FOR REMOTE LOGON
AFTACCT  DS        CL64     ACCOUNT FOR REMOTE LOGON
AFTTRACE DS        0F       TRACE FLAGS
         DS        XL3
*
* Note the following trace flags have been replaced
*      with TCPeep tracing - see Administrator Guide
*
AFTTRAC1 DS        XL1      TRACE FLAG 1
AFTNBRX  DS        F        NUMBER OF FILES TO BE TRANSFERED
AFTNBRXT DS        F        NUMBER OF FILES TRANSFERED
AFTRTNCD DS        F        RETURN CODE
AFTDGNCD DS        F        DIAGNOSTIC CODE
AFTRTNTA DS        A        RETURN TEXT - ADDRESS
AFTRTNTL DS        F        RETURN TEXT - LENGTH
*
AFTQTYPE DS        CL2      LOCAL FILE STORAGE QUEUE TYPE
AFTQNAME DS        CL8      LOCAL FILE STORAGE QUEUE NAME
AFTQITEM DS        F        TEMP STORAGE NUMBER OF ITEMS
*
AFTMODE  DS        CL1      FILE TRANSMISSION MODE
AFTMDFLT EQU       C' '     - USE DEFAULT MODE
AFTSTRM  EQU       C'S'     - STREAM MODE
AFTBLCK  EQU       C'B'     - BLOCK MODE (not implemented)
AFTCOMP  EQU       C'C'     - COMPRESSED MODE (not implemented)
*
AFTTYPE  DS        CL1      FILE TRANSMISSION TYPE
AFTTDFLT EQU       C' '     - USE DEFAULT TYPE
AFTASCII EQU       C'A'     - ASCII TRANSMISSION
AFTIMAGE EQU       C'I'     - IMAGE TRANSMISSION
AFTEBCDC EQU       C'E'     - EBCDIC TRANSMISSION (not implemented)
AFTLOCAL EQU       C'L'     - LOCAL TRANSMISSION (not implemented)
*
AFTFORM  DS        C        FILE TRANSMISSION FORMAT
AFTFDFLT EQU       C' '     - USE DEFAULT FORMAT
AFTNPRNT EQU       C'N'     - NON-PRINT
```

```
AFTTELNT EQU       C'T'     - TELNET FORMAT (not implemented)
AFTASACC EQU       C'A'     - ASA CARRIAGE CONTROL (not implemented)
AFTFOPTS DS        CL1      FILE TRANSMIT OPTIONS
AFTFBLNK EQU       C'B'     - STRIP TRAILING BLANKS
AFTTLIM  DS        F        TIME TO WAIT FOR CLIENT DATA
AFTRLIM  DS        H        RETRY LIMIT
*
         DS        CL1      AVAILABLE
AFTSTRU  DS        CL1      DATA STRUCTURE
AFTSDFLT EQU       C' '     - USE DEFAULT STRUCTURE
AFTFILE  EQU       C'F'     - FILE
AFTRECRD EQU       C'R'     - RECORD
AFTPAGE  EQU       C'P'     - PAGE (not implemented)
AFTALLO  DS        CL8      REMOTE STORAGE ALLOCATION SIZE
AFTSITEA DS        A        FTP SITE PARAMETERS - ADDRESS
AFTSITEL DS        F        FTP SITE PARAMETERS - LENGTH
AFTRNTOA DS        A        RENAME TO NAME - ADDRESS
AFTRNTOL DS        F        RENAME TO NAME - LENGTH
AFTWDIRA DS        A        REMOTE WORKING DIRECTORY - ADDR
AFTWDIRL DS        F        REMOTE WORKING DIRECTORY - LEN
AFTFNAMA DS        A        REMOTE FILE NAME - ADDRESS
AFTFNAML DS        F        REMOTE FILE NAME - LENGTH
AFTFUNC  DS        CL4      FTP SERVICE COMMAND
AFTFTPCD DS        F        FTP RETURN CODE
AFTFTPTA DS        A        FTP RETURN TEXT - ADDRESS
AFTFTPTL DS        F        FTP RETURN TEXT - LENGTH
AFTLEN   EQU       *-CFCAFT LENGTH OF AFT
```

AXL: Argument for Data Translation Used by the Translate API Service

This section describes the Unicenter SOLVE:CPT Argument for data translation, the AXL. The AXL is used by the TRANSLATE service to define the arguments to translate.

The following information is provided:

- An offset table of the AXL fields
- An alphabetical list of AXL fields
- A sample copy of the T09DAXL DSECT

Note: The AXL control block is 32 bytes in length, which is x'20' in hexadecimal. The space for this control block must be created by the application and mapped to by the sample T09DAXL DSECT.

Offsets

This table provides information from the T09DAXL DSECT member with field descriptions.

Decimal	Hex	Type	Length	Name	Description
0	(0)			AXL	Translate parameters.
0	(0)	HALF WORD	2	AXLVERS	Version number.
2	(2)	HALF WORD	2	AXLFUNC	Function code.
4	(4)	ADDRESS	4	AXLTOKEN	Token (CEP).
8	(8)	ADDRESS	4	AXLSADDR	Source text address.
12	(C)	FULL WORD	4	AXLSLENG	Source text length.
16	(10)	FULL WORD	4	AXLRTNCD	Return code.
20	(14)	FULL WORD	4	AXLDGNCD	Diagnostic code.
24	(18)	HALF WORD	2	AXLXMODE	Character set mode.
24	(18)	BYTE	1	AXLXMOD2	Mode 2.

Decimal	Hex	Type	Length	Name	Description
25	(19)	BYTE		AXLXMOD1	Mode 1.
		....		AXLMSBCS	- Single-byte character set.
		1		AXLMDBCS	- Double-byte character set.
		.... ..1.		AXLMMIXD	- Mixed SBCS/DBCS character set.
		.... .1..		AXLMNUMS	- Numeric character set.
26	(1A)	HALF WORD	2	AXLLTYPE	Translation type request.
26	(1A)	BYTE	1	AXLLTYP2	Type 2.
27	(1B)	BYTE	1	AXLLTYP1	Type 1.
		1		AXLTATOE	- Translate ASCII to EBCDIC.
		.... ..1.		AXLTETOA	- Translate EBCDIC to ASCII.
		.... .1..		AXLTAUPC	- Translate ASCII to uppercase.
		.... 1...		AXLTEUPC	- Translate EBCDIC to uppercase.
28	(1C)	ADDRESS	4	AXLTABLE	Address of user translation table.
32	(20)			AXLLEN	Length of AXL.

Alphabetized Cross-Reference Table

This table provides an alphabetized list of field names mapped to the offset within the AXL control block.

Name	Hex Offset	Hex Value
AXLDGNCD	14	
AXLFUNC	2	
AXLLEN	20	20
AXLLTYPE	1A	
AXLLTYP1	1B	
AXLLTYP2	1A	
AXLMDBCS	19	01
AXLMMIXD	19	02
AXLMNUMS	19	04
AXLMSBCS	19	00
AXLRTNCD	10	
AXLSADDR	8	
AXLSLENG	C	
AXLTABLE	1C	
AXLTATOE	1B	01
AXLTAUPC	1B	04
AXLTETOA	1B	02
AXLTEUPC	1B	08
AXLTOKEN	4	
AXLVERS	00	
AXLXMOD2	18	
AXLXMODE	18	

Sample DSECT Member T09DAXL

The following is a sample of DSECT member T09DAXL located in the ***cpthlq.***T09MAC distributed library.

It:

- Contains the layout and field names for your use in your application program
- Is used by all the provided application programming samples

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

```
Name     Operation Operands Description

AXL      DSECT              CPT ARGUMENT FOR TRANSLATION
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
AXLMSBCS EQU       X’00’    * SINGLE BYTE CHARACTER SET
AXLMDBCS EQU       X’01’    * DOUBLE BYTE CHARACTER SET
AXLMMIXD 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 TRANSLATION TABLE
AXLLEN   EQU       *-AXL    LENGTH OF AXL
```

Client Data Listener Transaction Start

The transaction that is initiated by the Client Data Listener Tool is passed this control block. This structure is accessed by through an EXEC CICS RETREIVE command in the invoked (spawned child) transaction.

For further information refer to Client/Data Listener Option contained in the section: T09MLSTN Macro in the "Configuration Reference" chapter of the *Administrator Guide.*

The following information is provided: A sample copy of the T09DCSKL DSECT.

Note: The Client Data Listener control block is 72 bytes in length, which is x'48' in hexadecimal. The storage for this control block will be allocated by CICS when the EXEC CICS RETREIVE command is issued. The storage should then be mapped to by the sample T09DCSKL DSECT.

Sample DSECT Member T09DCSKL

The following is a sample of DSECT member T09DCSKL located in the ***cpthlq.***T09MAC distributed library.

It:

- Contains the layout and field names for your use in your application program
- Is used by all the provided application programming samples

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

```
Name     Operation Operands Description

CSK      DSECT              CPT Client Data Listener parmlist
CSKCPARM DS        0F
CSKTOKEN DS        F        New token - socket ID
CSKLNAME DS        CL8      Listener name
CSKLSUBN DS        CL8      Listener subname
CSKCDATA DS        CL35     Up to 35 bytes of client data
         DS        CL1      C language delimiter
         DS        0F
CSKDMN   DS        H        Family
CSKRPORT DS        H        Remote port
CSKRADDR DS        F        Remote IPADDR
         DS        XL8'00'  Reserved
CSKLEN   EQU       *-CSKCPARM
```

Connection Time Security Program Control Block

When security is turned on in the T09MCICS or T09MLSTN macros in the configuration file, control is passed to the specified security program. The security program is passed this control block. This structure is accessed by through an EXEC CICS RETREIVE command in the invoked (spawned child) transaction.

The following information is provided:

- An offset table of the security program fields
- A sample copy of the T09DSEC DSECT

Note: The security program control block is 596 bytes in length, which is x'254' in hexadecimal. The storage for this control block will be allocated by CICS when the EXEC CICS RETREIVE command is issued. The sample T09DSCTY DSECT should then be mapped to the storage.

For further information, see About the Optional Security Program in the "Security" appendix of the *Administrator Guide.*

Offsets

This table provides information from the T09DSCTY DSECT member with field descriptions.

Decimal	Hex	Format	Field	Description
0	(0)	4-byte character	SECTRAN	Requested server transaction, maybe modified by the program.
4	(4)	40-byte character	SECDATA	Client data, if available.
44	(2C)	2-byte character	SECSTRT	Method of server initiation: KC, TC, or IC.
46	(2E)	6-byte character	SECICTM	IC Hours, Minutes, Seconds.
52	(34)	2-byte binary	SECAFAM	Address family: Inet domain=2.
54	(36)	2-byte binary	SECRPRT	Client remote port number.
56	(38)	4-byte binary	SECRHST	Client remote host IP address.

Decimal	Hex	Format	Field	Description
60	(3C)	1-byte character	SECACTN	Authorization switch: ■ 1=accept ■ 0=fail
61	(3D)	4-byte character	SECTMID	Associated terminal facility.
65	(41)	2-byte binary	SECLPRT	Requested server local port.
67	(43)	8-byte binary	SECUSER	Returned user ID
75	(4B)	4-byte binary	SECTOKN	Token that represents the TCP connection.
79	(4F)	4-byte binary	SECLHST	Local host IP address.

Sample DSECT Member T09DSCTY

The following is a sample of DSECT member T09DSCTY located in the ***cpthlq.***T09MAC distributed library.

It:

- Contains the layout and field names for your use in your application program
- Is used by all the provided application programming samples

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

```
Name     Operation Operands Description

SECPARM  DSECT              CPT Security parmlist
SECPARM  DS        0A       CPT GLOBAL PARAMETERS
SECTRAN  DS        CL4      SERVER TRANSACTION REQUESTED
SECDATA  DS        XL40     REQUESTOR DATA
SECSTRT  DS        CL2      HOW TASK IS TO BE STARTED
SECICTM  DS        XL6      INTERVAL CONTROL TIME
SECADRS  DS        0CL8     REQUESTOR ADDRESS
SECAFAM  DS        H        DOMAIN
SECRPRT  DS        H        PORT
SECRHST  DS        F        HOST IP ADDRESS
SECACTN  DS        CL1      PERMIT/PROHIBIT SWITCH
SECPRMT  EQU       C'1'     ..OKAY, INITIATE TASK
         DS        X        RESERVED
SECTMID  DS        CL4      ANY ASSOCIATED CICS TERMINAL
SECLPRT  DS        H        LOCAL SERVER PORT
SECUSER  DS        CL8      USER ID
         DS        CL512    RESERVED
SECTOKN  DS        F        TOKEN - ENDPOINT
SECLHST  DS        F        LOCAL HOST
SECLEN   EQU       *-SECPARM  LENGTH OF SECURITY DATA AREA
```

Parameter List Passed to T09MTRAN Initiated Transactions

Important! *There are two formats for the parameter list that is passed to the transaction initiated by Unicenter SOLVE:CPT when configured via the T09MTRAN. The simple format is when the transaction is passed the string of data that is contained within quotes of the PARMDATA parameter of the T09MTRAN macro.*

For complete details and samples on how to have Unicenter SOLVE:CPT initiate your CICS transaction for you, see the "T09MTRAN Programming Notes" appendix in this guide.

The T09MTRAN macro is part of the configuration table that is fully documented in the *Administrator Guide*.

The second format of is the list parameter (LSTP) layout. The LSTP format is used whenever any of the CFG0000 fields (fields other than USERID, TERMID, TRANSID and PARMDATA) are configured from the T09MTRAN macro. This layout consists of 64 bytes passed to the application as shown below.

T09DLST DSECT Sample

The following is a sample of DSECT member T09DLST located in the ***cpthlq.***T09MAC distributed library.

```
LSTID    DS    CL4'LSTP'   ID for Control block
LSTLEN   DS    H           Total length of LSTPARMS DSECT
LSTVERS  DS    XL2         Version number
LSTOPTDA DS    A           Pointer to the optional Data
LSTOPTLN DS    F           Length  of the optional Data
LSTCFGDA DS    A           pointer to the CFG0000 field
LSTCFGLN DS    F           Length  of the CFG0000 field
LSTLCADA DS    A           Pointer to the current LCA
LSTLCALN DS    F           Length  of the current LCA
LSTRESVD DS    XL32        Reserved for future use
```

A program can specify the LSTP DSECT by adding the following line to an Assembler program:

```
T09DLST MF=DSECT
```

Field Descriptions

LSTID	Tag identification of 'LSTP'
LSTLEN	Length field of the LSTP DSECT.
LSTVERS	Current version number of the DSECT.
LSTOPTDA	Pointer to a copy of the data specified on the PARM field in the T09MTRAN entry in the T09CON*xx* configuration file.
LSTOPTLN	Length of the LSTOPTDA field.
LSTCFGDA	Pointer to a copy of the data specified on the CFG0000 fields in the T09MTRAN entry in the T09CON*xx* configuration file.
LSTCFGLN	Length of the LSTCFGDA field.
LSTLCADA	A pointer to the LCA entry for this transaction.
LSTLCALN	Length of the LSTLCADA field.

LCA0000 and CFG0000 Control Blocks

To be run time compatible with IBM's CICS sockets some of the control block architecture is provided so that programs written to run in the IBM CICS sockets environment execute transparently in a Unicenter SOLVE:CPT environment.

Two control block structures LCA0000 and CFG0000 that are created as part of IBM's EZACONFIG configuration file are also created for Unicenter SOLVE:CPT. See IBM's *Communications Server IP CICS Sockets Guide* for full details on how to use these features.

The LCA0000 control block can be expanded in an Assembler program by the following macro expansion:

```
EZACICA AREA=LCA,TYPE=DSECT
```

The CFG0000 control block can be expanded in an Assembler program by the following macro expansion:

```
EZACICA AREA=CONFIG,TYPE=DSECT
```

The following Assembler listings are provided for your convenience and are not intended to replace the IBM documentation in this area.

LCA0000 Control Block Listing

```
          EZACICA AREA=LCA,TYPE=DSECT
+*
+*        DSECT FOR LISTENER CONTROL AREA
+*
+LCA0000  DSECT
+*
+LCATECB  DS    F                  Termination ECB
+LCATRAN  DS    CL4                Name of Listener transaction
+LCASTAT  DS    X                  Status of this listener
+LCASTAT0 EQU   B'00000000'        Listener not in operation
+LCASTATI EQU   B'00000001'        Listener in initialization
+LCASTATS EQU   B'00000010'        Listener in SELECT
+LCASTATP EQU   B'00000100'        Listener processing
+LCASTATE EQU   B'00001000'        Listener had initialization error
+LCASTATC EQU   B'00010000'        Immediate termination in progress
+LCASTATD EQU   B'00100000'        Deferred termination in progress
+LCAPHASE DS    X                  Execution phase for IBM listener
+LCAEND   DS    0F                 Alignment
+LCALEN   EQU   LCAEND-LCA0000     Length of Listener Control Area
+LCACHAIN DS    A                  Address of next LCA on chain
+LCAEND2  DS    0F                 Alignment
+LCALEN2  EQU   LCAEND2-LCA0000    Length of chained LCA
```

CFG0000 Control Block Listing

```
,,        EZACICA AREA=CONFIG,TYPE=DSECT
+*
+*        DSECT FOR CICS/SOCKETS CONFIGURATION FILE
+*
+CFG0000  DSECT
+CFHAPPL  DS    CL8                APPLID of CICS
+*
+CFHRTYPE DS    CL1                Record Type
+CFHRTYPC EQU   C'C'               CICS Record
+CFHRTYPL EQU   C'L'               Listener Record
+*
+         DS    XL3                Reserved
+*
+*        Record Layout for CICS Record
+*
+CFCTRAN  DS    XL4                Binary Zeros
+CFCTCPIP DS    CL8                Address Space Name for TCP/IP
+CFCNOTSK DS    H                  Number of Reusable tasks
+CFCSTIME DS    H                  Cache Minimum Refresh Time
+CFCLTIME DS    H                  Cache Maximum Refresh Time
+CFCNORES DS    H                  Cache Number of Concurrent Resolver
+CFCDPRTY DS    H                  Limit Priority of Subtask
+CFCENAME DS    CL4                Name of TD Error Queue
+CFCOPT   DS    X                  CICS Options                  @L
+CFCOPTSS EQU   B'00000001'        Suppress task started messages   @L
+*
```

```
+*        Record Layout for Listener Record
+*
+         ORG   CFCTRAN            Reset Location Counter
+CFLTRAN  DS    CL4                Listener Transaction Name
+CFLPORT  DS    H                  Port Number for Listener
+CFLBKLOG DS    H                  Backlog value for Listener
+CFLNSOCK DS    H                  Number of Sockets Used by Listener
+CFLNMIN  DS    H                  Minimum Length of Input Message
+CFLLTIME DS    H                  Timeout value in seconds for accept
+CFLRTIME DS    H                  Timeout value in seconds for read
+CFLGTIME DS    H                  Timeout value in seconds for
+                                  givesocket
+CFLOPT   DS    X                  Listener Options
+*
+CFLOPTIS EQU   B'00000001'        Immediate Startup
+CFLOPTTE EQU   B'00000110'        Translate entire message
+CFLOPTTR EQU   B'00000010'        Translate Transaction Code Only
+CFLOPTUD EQU   B'00000100'        Translate User Data Only
+*              B'00001000'        Reserved for compatibility
+CFLOPTPD EQU   B'00010000'        Peek at data only
+CFLOPTEB EQU   B'00100000'        Outbound messages are in EBCDIC
+CFLOPTEL EQU   B'01000000'        This is an ENHANCED listener
+*
+CFLSECXT DS    CL8                Name of security exit
+CFLWLMN1 DS    CL12               WLM group name 1            @A1
+CFLWLMN2 DS    CL12               WLM group name 2            @A1
+CFLWLMN3 DS    CL12               WLM group name 3            @A1
+CFLCSTRN DS    CL4                Child server tranid
+CFLCSSTT DS    CL2                Child server startup type
+CFLCSDLY DS    CL6                Child server delay interval
+CFLMSGLN DS    H                  Length of inbound message
+         ORG   CFG0000+150        Leave some reserved area
+*
+CFGLEN   EQU   *-CFG0000          Length of record
```

Appendix

C

T09MTRAN Programming Notes

This chapter provides additional information on programming concerns when using the T09MTRAN Unicenter SOLVE:CPT configuration table statement.

The T09MTRAN macro defines a CICS transaction that can be started by Unicenter SOLVE:CPT. It is an excellent mechanism to start non-T09MLSTN servers after Unicenter Solve: CPT is properly initialized. There is no requirement that the transaction be a server, any transaction start that relates to Unicenter SOLVE:CPT can be managed by the T09MTRAN configuration macro.

For full details on configuration of the T09MTRAN macro, see the chapter "Configuration Reference" in the *Administration Guide*.

Operationally you can use the T09MTRAN macro startup two ways:

- Using the default parameter of IMMED=YES, the transaction is automatically started immediately after Unicenter SOLVE:CPT completes proper startup initialization
- If you code IMMED=NO, the you can manually start the transaction anytime after Unicenter SOLVE:CPT initialization completes

For information and sample use of the IPUL transaction to start an IMMED=NO defined transaction, see the chapter "Operations" in the *Administration Guide*.

This appendix discusses the following topics:

- Parameter List Passed to T09MTRAN Initiated Transactions
- LCA0000 and CFG0000 Control Block Programming Notes

Parameter List Passed to T09MTRAN Initiated Transactions

There are two possible layouts of data passed to the customer's transaction:

- Basic layout where the data string is simply passed from the PARM field of the T09MTRAN configuration table macro. *This is the recommended default*.
- EZACONFG layout The LSTP DSECT with a length of 64 bytes.

Regardless of the parameter list the application issues a one of the following CICS retrieve call to access the data:

```
EXEC  CICS RETRIEVE INTO() LENGTH()
```

or

```
EXEC  CICS RETRIEVE SET() LENGTH()
```

Basic Layout: Data Passed from the PARM Field

When using the basic layout, data placed on the PARM field is passed to the T09MTRAN transaction. The transaction can retrieve the PARM data by issuing the "EXEC CICS RETRIEVE ..." command.

This layout applies anytime that a user listener application has *not* configured any of the CFG0000 fields in the T09MTRAN macro.

In this case the only fields that can be coded in the T09MTRAN macro are:

- APPLID
- ID
- PARM
- TERMID
- TRANSID
- USERID

Basic Layout Usage Examples

Using the sample examples as described in the "Operations" chapter of the *Administration Guide*:

```
T09MTRAN TRANSID=SRV1,PARM=1344
T09MTRAN TRANSID=SRV2
T09MTRAN TRANSID=SRV3,PARM='1346,IP=138.141.222.17',ID=ID1346
T09MTRAN TRANSID=SRV3,PARM=1347,ID=ID1347
```

The following start scenarios occur at CPT initialization:

- Start of transaction SRV1 will start with '1344' passed as character data.

  Note: A very common need is to pass the PORT parameter, as shown in this example of passing the server port number of 1344. For programmers that prefer to avoid using pointers, this method of passing the port may be easier than using the EZACONFG layout which requires pointer use.

- Start of transaction SRV2 where no data is passed to the application
- Start of transaction SRV3 with a character string of '1346,IP=138.141.222.17' being passed in the common area
- Start of transaction SRV3 will start with '1347' passed as character data

EZACONFG Layout: LSTP DSECT (64 Byte Length)

The LSTP parameter structure is passed as PARM data whenever any CFG0000 related parameters are specified on the T09MTRAN macro. The transaction can retrieve the LSTP data by issuing the "EXEC CICS RETRIEVE ..." command.

The CFG0000 parameters that cause a LSTP parmlist to be passed to the T09MTRAN transaction are listed below.

- PORT *(see note below)*
- BACKLOG
- ACCTIME
- REATIME
- GIVTIME
- NUMSOCK
- MINMSGL
- TRANTRN
- TRANUSR
- SECEXIT
- WLMGN1
- WLMGN2
- WLMGN3

LSTP DSECT layout is used whenever any of the CFG0000 fields (fields other than USERID, TERMID, TRANSID and PARMDATA) are configured from the T09MTRAN macro.

A program can specify the LSTP DSECT by adding the following line to an Assembler program:

```
T09DLSTP MF=DSECT
```

For full details on the LSTP DSECT, the LCA0000, and CFG0000 fields and layouts please refer to the appendix “Control Block Layouts.”

EZACONFG Layout Usage Example

```
T09MTRAN TRANSID=SRV4,PORT=4444,BACKLOG=15,WLMGN1=CPTGRP
```

Any parameter from the list above will create the EZACONFG format of parameter list.

The transaction SRV4 is started and the LSTP DSECT is passed parameter format.

LCA0000 and CFG0000 Control Block Programming Notes

To be runtime compatible with IBM's CICS sockets, some of the control block architecture is provided so that program's written to run in the IBM CICS sockets environment will execute transparently in a Unicenter SOLVE:CPT environment.

Two control block structures LCA0000 and CFG0000 are created as part of IBM's EZACONFG configuration file are also created for Unicenter SOLVE:CPT. See IBM's *Communications Server IP CICS Sockets Guide* for full details on how to use these features.

Each T09MTRAN entry must be unique by the TRANSID parameter when it wants to participate in the LCA array field. There is no ID field available inside the LCA DSECT. There is no way to uniquely identify two or more transactions using the same transaction ID parameter TRANSID. This is not a severe limitation within CICS. Multiple transactions can point to the same program. So, a site needs to uniquely identify each server transaction name in the TRANSID parameter for each T09MTRAN entry in the T09CON*xx* configuration file. The ID field will always be unique among all T09MTRAN entries.

For full details on the LSTP DSECT, the LCA0000, and CFG0000 fields and layouts please refer to the appendix "Control Block Layouts" in the guide.

The LCA0000 control block can be expanded in an Assembler program by the following macro expansion:

```
EZACICA AREA=LCA,TYPE=DSECT
```

An application can use the EXTRACT EXIT command for program EZACIC01 to find the global work area pointer:

```
EXEC CICS EXTRACT EXIT
PROGRAM(EZACIC01)
GASET(ptr)GALEN(len)
```

At offset zero of the global work area (GWA) will be the literal "ACDC."

At offset x'170 of the GWA will be a pointer to an array of 12 byte LCA entries. An application can walk through the array until it finds a LCA entry where the LCATRAN transaction ID matches its own server transaction ID.

The CFG0000 control block can be expanded in an Assembler program by the following macro expansion:

```
EZACICA AREA=CONFIG,TYPE=DSECT
```

A program can specify the LSTP DSECT by adding the following line to an Assembler program:

```
T09DLSTP MF=DSECT
```

Appendix D

Linking EZASOKET, EZACICSO, and EZACICAL Applications

The IBM CICS Sockets feature of CICS provides a callable application program interface (API). The API is supported by all CICS supported languages like Assembler, COBOL, C, and PL/I. This style of TCP/IP sockets API support is exemplified by using calls to modules EZASOKET, EZACICSO, or EZACICAL in your CICS application program. The EZASOKET, EZACICSO, or EZACICAL CICS applications use stub programs to provide TCP/IP sockets API support. These stubs are part of a CICS TRUE (Task Related User Exit) because they use external MVS services. Applications must pull in these API at the end of the link-edit step of their compile-and-link JCL statements to provide support for the API calls. The mentioned API's are referred to as the IBM CICS Sockets API.

Do not get confused with EZASOKET, EZACICSO and EZACICAL socket applications or IBM CICS Sockets API applications with CPT applications.

Stubs for the following applications are as given below:

- EZASOKET socket applications must link in stub named EZACICAL, which has an EZASOKET entry point
- EZACICSO socket applications must link in stub named EZACICSO
- EZACICAL socket applications must link in stub named EZACICAL
- Unicenter SOLVE:CPT applications must link in stubs starting with T09F*

The following topics are discussed in this appendix:

- Linking EZASOKET Applications
- Linking EZACICSO Applications
- Linking EZACICAL Applications

Linking EZASOKET Applications

The CICS applications that do source code calls to EZASOKET must link editor step and include the EZACICAL module. The EZACICAL module contains an EZASOKET entry point. Performing the link enables the API stub for CICS applications that call the EZASOKET module.

For an example of the linkage editor control cards needed to include the EZACICAL module, see Example: Linkage Editor Control Cards in this chapter. You must include the EZACICAL module from either the IBM Communications Server IP load library (*tcpiphlq*.SEZATCP) or the Unicenter SOLVE:CPT 6.1 distribution library (*cpthlq*.T09LOAD). EZACICAL has an alias of T09ESOKT in the *cpthlq*.T09LOAD library.

Important! *You cannot link-edit and include the EZASOKET module directly. It causes unpredictable results. The real EZASOKET module is a batch only TCP/IP socket interface that directly calls the EZASMI API. Linking in the EZASOKET module causes the EZASMI API calls to be made in the CICS region, which exposes the CICS quasi-reentrant (QR) TCB to many MVS system waits. It can put a CICS region in a hanging wait state. Avoid linking in the EZASOKET module directly to your CICS application program.* ***To avoid the confusion of linking EZASOKET module, you can consider using the EZACICSO API calls instead.***

Example: Linkage Editor Control Cards

```
//  JOB
....
//LKED     EXEC PGM=IEWL....
....
//SYSLIB   DD DISP=SHR,DSN=cicsHLQ.SDFHLOAD
//T09LOAD  DD DISP=SHR,DSN=CPTHLQ.T09LOAD    CA's  EZASOKET
//MYOBJLIB DD DISP=SHR,DSN=myobjlib
//SYSLIN   DD *
 INCLUDE  MYOBJLIB(myprogname)
 INCLUDE  T09LOAD(EZACICAL)
 INCLUDE  SYSLIB(DFHEAI)
 INCLUDE  SYSLIB(DFHEAI0)
 MODE     AMODE(31),RMODE(24)
 ENTRY    DFHEAI
 ORDER    DFHEAI,myprogname
 NAME     myprogname(R)
```

Example: Link Control Cards for IBM TCP Previously Compiled Programs

Here is an example of how to link your EZASOKET module when using the IBM CICS Sockets feature.

Note: The only change in the example is for the ddname on the INCLUDE control card that points to a different load library to pick up another EZACICAL module. You should not re-link any CICS applications that are currently running over the IBM CICS Sockets feature when they migrate to run over CPT.

```
//  JOB
....
//LKED      EXEC PGM=IEWL....
....
//SYSLIB    DD DISP=SHR,DSN=cicsHLQ.SDFHLOAD
//IBMEZA    DD DISP=SHR,DSN=TCPIP.SEZATCP     IBM's EZASOKET
//MYOBJLIB DD DISP=SHR,DSN=myobjlib
//SYSLIN    DD *
 INCLUDE   MYOBJLIB(myprogname)
 INCLUDE   IBMEZA(EZACICAL)
 INCLUDE   SYSLIB(DFHEAI)
 INCLUDE   SYSLIB(DFHEAI0)
 MODE      AMODE(31),RMODE(24)
 ENTRY     DFHEAI
 ORDER     DFHEAI,myprogname
 NAME      myprogname(R)
```

Linking EZACICSO Applications

For the z/OS V1R4 version of Communications Server, IBM provides a mechanism around some of the confusing points of linking EZASOKET API CICSmodule The EZACICAL API is not used until z/OS V1R4, yet when you link the recommended EZASOKET API, you had to link-edit to include the non-recommended EZACICAL module. This concept causes more confusion on higher level language linkage editor (Binder) steps in JCL.

The solution to this linking problem has been provided with the EZACICSO TCP/IP sockets API. When using the EZACICSO API, you can use the same module name on the program source code CALL statement as that used in the JCL linkage editor step. Using the EZACICSO API lets you use the Binder's Automatic Library Call option (AUTOCALL) to build the load modules if you want to use the higher level language.

You must include EZACICSO from either the IBM Communications Server IP load library (*tcpiphlq*.SEZATCP) or the Unicenter SOLVE:CPT 6.1 distribution library (*cpthlq*.T09LOAD). EZACICSO has an alias of T09ESOKT in the *cpthlq*.T09LOAD library.

Example: EZACICSO Linkage Editor Control Cards

```
//  JOB
....
//LKED      EXEC PGM=IEWL....
....
//SYSLIB    DD DISP=SHR,DSN=cicsHLQ.SDFHLOAD
//T09LOAD   DD DISP=SHR,DSN=CPTHLQ.T09LOAD    CA's  EZASOKET
//MYOBJLIB DD DISP=SHR,DSN=myobjlib
//SYSLIN    DD *
 INCLUDE  MYOBJLIB(myprogname)
 INCLUDE  T09LOAD(EZACICSO)
 INCLUDE  SYSLIB(DFHEAI)
 INCLUDE  SYSLIB(DFHEAI0)
 MODE     AMODE(31),RMODE(24)
 ENTRY    DFHEAI
 ORDER    DFHEAI,myprogname
 NAME     myprogname(R)
```

Linking EZACICAL Applications

The EZACICAL API was delivered with IBM TCP V2.2.1. It was replaced with the EZASOKET API that was delivered with IBM TCP V3.1.

Important! *This API is supported for downward compatibility only. You should use the EZASOKET or EZACICSO API when building new applications.*

The EZACICAL API is required when linking CICS applications that use the EZACICAL API.

You must include EZACICAL from either the IBM Communications Server IP load library (*tcpiphlq*.SEZATCP) or the Unicenter SOLVE:CPT 6.1 distribution library (*cpthlq*.T09LOAD). EZACICAL has an alias of T09ESOKT in the *cpthlq*.T09LOAD library.

Appendix

E

T09MCALL Macro

This chapter provides information about the T09MCALL macro used in all the Assembler programming samples.

The macro provides two advantages:

- Easier and more readable call format
- Setting of the version number

  Setting the version number eliminates the need to update your code. If the version number ever changes, a simple recompile enables your code for the new version.

Important! *We recommend always using the T09MCALL macro.*

Macro Syntax

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]
```

service

T09MCALL can call any of the Unicenter SOLVE:CPT services:

CLOSE	Keyword specifying the API CLOSE service.
CONNECT	Keyword specifying the API CONNECT service (TCP only).
GIVE	Keyword specifying the API GIVE service.
LISTEN	Keyword specifying the API LISTEN service (TCP only).

RCVFROM	Keyword specifying the API RCVFROM service (UDP only).
RECEIVE	Keyword specifying the API RECEIVE service (TCP only).
SELECT	Keyword specifying the API SELECT service.
SEND	Keyword specifying the API SEND service (TCP only).
SENDTO	Keyword specifying the API SENDTO service (UDP only).
TAKE	Keyword specifying the API TAKE service.
TRANS	Keyword specifying the API TRANSLATE service.

PARM *service argument* Name of storage area of length four bytes. This storage area contains the address of a storage area containing the service specific request argument.

For example for a CLOSE call, this storage location specified by PARM contains the address of your CPT ACL(Argument for Close) control block. The address of the argument is loaded into register 1.

This field is optional and the argument is assumed to be in register 1. The macro updates the high order bit of the storage location specified by the PARM parameter to indicate the last parameter in a standard MVS parameter list with only one parameter.

VERSION *argument version number*

Specifies the Unicenter SOLVE: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.

Usage Example

We recommend that a site use the T09MCALL macro to make a CPT call because it fills in the correct version number in the passed parameter list.

Here is an example using T09MCALL to call RECEIVE:

```
         T09DADT MF=DSECT         Argument for RECV

CPTPARMS DS    F                  CPT calling paramater
RECVARG  DS    XL(ADTLEN)         Argument for RECV
...
         LA    R01,RECVARG        Create a pointer to your ADT
         ST    R01,CPTPARMS       Store the pointer in your parmlist area
         T09MCALL RECEIVE,PARM=CPTPARMS    Call CPT receive service
```

Index

A

abortive close example, 3–4

ACL
  sample DSECT member T09KDACL, B–4

ACL control block
  alphabetized field name cross reference table, B–3
  offsets, B–2

ACM
  sample DSECT member T09KDACM, B–8

ACM control block
  alphabetized field name cross reference table, B–7
  offsets, B–5

ADT
  control block, B–10

ADT control block
  alphabetized field name cross reference table, B–11
  offsets, B–10

ADT control bock
  sample DSECT member T09KCADT, B–12

AFM control block
  alphabetized field name cross reference table, B–15
  offsets, B–14
  sample DSECT member T09DRTCD, B–18
  sample DSECT member T09KDAFM, B–16

AFT
  AFTFTPTX, 5–9
  AFTRTNTX, 5–9

AFT control block
  alphabetized cross reference table, B–17
  offsets, B–17

AFTFTPTX of the AFT, 5–9

AFTRTNTX of the AFT, 5–9

API services, 1–2

applications
  linking
    EZACICAL, D–3, D–4
    EZASOKET, D–1, D–2

argument
  for CLose, B–2

argument for
  connection management, B–5
  data transfer, B–10
  data translation, B–17, B–20
  facility management, B–14

arguments for
  CLOSE (ACL)
    service, 2–11
  connection management (ACM), 2–5
  data transfer (ADT), 2–8
  facility management (AFM), 2–14
  translation (AXL), 2–12

ASCII data
  translating within a user buffer, 2–12

Assembler DSECTs
  CONNECT service, 4–4
  GIVE service, 6–4
  LISTEN service, 7–9
  RECEIVE service, 9–34
  RECVFROM service, 8–6
  SEND service, 10–9
  SENDTO service, 11–5
  TAKE service, 12–4
  TRANSLATE service, 13–6

Assembler DSECTS
  CLOSE service, 3–5

Assembler macro T09DRTCD, A–4

automated Unicenter SOLVE:CPT transaction processing options, 1–3

AXL control block
alphabetized cross reference table, B–22
offsets, B–20
sample DSECT member T09KDAXL, B–23

AXLTOKEN, TRANSLATE service
token, 13–9

AXLVERS version number, TRANSLATE service

B

blocking LL RECEIVE, 9–26

blocking RECEIVE loop, 9–32

blocking separator character RECEIVE, 9–28

blocking, RECEIVE service, 9–3

C

CA-InterTest, using with Unicenter SOLVE:CPT aplications, 2–39

call syntax
CLOSE service, 3–2
FTP Client service, 5–2
GIVE service, 6–2
RCVFROM service, 8–3
RECEIVE service, 9–2
SEND service, 10–2
SENDTO service, 11–3
TRANSLATE service, 13–2

CICS
API services, 3–1
Unicenter SOLVE:CPT
code tools, 1–3

client callable service, 1–2

client condensed pseudo code, 2–4

client/data listener with translation, 7–26

client/server
applications code examples, 2–23
mechanisms for TCP data transfer, 2–7

client-data listener option. *See* Listener

client-data option data atructure, 7–26

CLOSE service
arguments for CLOSE (ACL), 2–11
Assembler DSECTS, 3–5
complete parameter list, 3–11, 3–12
completion of, 2–11
connection and endpoint release, 2–11
defined, 3–1
examples
abortive close, 3–4
graceful close, 3–3
invoking, 3–1
parameter values returned in the ACL, 3–5
processing control options
ACLABORT, 3–10
ACLORDER, 3–10
recommended ACL parameters, 3–2
sample programs, 3–6
syntax, 3–2
termination, 3–9
abortive, 3–7
graceful, 3–7
token (ACLTOKEN), 3–9
usage, 3–3
usage notes, 3–9
using the ACL, 3–9
version number (ACLVERS), 3–9

COBOL
SOLVE:CPT API
to invoke the Listener with translation from, 7–27

code examples
client/server applications, 2–23
server application, 2–25
multithreaded CICS, 2–27, 2–31
multithreaded data processing, 2–29

compiling and linking
CPT/API applications. *See the "Unicenter SOLVE:CPT API Services chapter.*, *See the "Unicenter SOLVE:CPT API Services chapter.*
EZACICAL applications, D–3, D–4
EZASOKET applications, D–2
IBM CICS sockets applications, D–1

complete parameter list
CLOSE service, 3–12
CONNECT service, 4–15
FTP Client service, 5–14, 5–20
GIVE service, 6–7
LISTEN service, 7–23
RCVFROM service, 8–15
RECEIVE service, 9–47
SELECT service, 10–19
SENDTO service, 11–14
service, 11–11

TAKE service, 12–9
TRANSLATE service, 13–10, 13–11

complete parameter lists
CLOSE service, 3–11
CONNECT service, 4–10
LISTEN service, 7–18
RCVFROM service, 8–10
RECEIVE service, 9–41
SEND service:, 10–15
SENDTO service, 11–9
TAKE service, 12–9

completion information
CLOSE service, 3–7
CONNECT service, 4–5
FTP Client service, 5–8
GIVE service, 6–5
LISTEN service, 7–11
RECEIVE service, 9–36
SEND service, 10–10
TAKE service, 12–5
TRANSLATE service, 13–7
when the ACMTRNID field is not specified in the CPT-ACM, 7–12

CONNECT service, 2–6
arguments, 4–2, 7–2
Assembler DSECTs, 4–4
complete parameter list, 4–10, 4–15
completion information, 4–5
creating connections, 2–13
data transfer buffering, 4–9
defined, 4–1
established connection information, 4–8
example, 4–3
implementing TCP/IPfacilities, 4–8
invoking, 4–1
parameter values returned in the ACM, 4–4
recommended ACM parameters, 4–2
remote
host name (ACMRNAME), 4–8
IP address (ACMRADDR), 4–8
required and optional fields, 4–8
return codes, 3–8, 4–6
sample programs, 4–5
service name (ACMSRVCE), 4–8
syntax, 4–2
transport provider port number (ACMPORT), 4–8
updating the ACM, 4–5
usage notes, 4–8
version number (ACMVERS), 4–8

connection management
establishing connections, 2–5

connection release, 2–13
CLOSE service, 2–11
TRUE management routines, 2–11

connection time security program control block, B–25

connections, creating
CONNECT service, 2–13
LISTEN service, 2–13

control block layouts, B–1

control blocks
connection time security program, B–25
sample DSECT member DSCTY, B–26
sample DSECT member T09DCSKL, B–24

control blocks
ACL, B–2
cross reference, B–3
offsets, B–2
ACM
offsets, B–5
ACM alphabetized field name
cross reference, B–7
ADT
offsets, B–10
AFM, B–14
offsets, B–14
AFT, B–17
offsets, B–17
alphabetized AFT
cross reference, B–17
alphabetized AXL
cross reference, B–22
alphabetized field name ADT
cross reference, B–11
alphabetized field name AFM cross reference, B–15
argument for ACM, B–5
argument for ADT, B–10
AXL, B–20
offsets, B–20

control blocks
LCA0000, B–29

control blocks
LCFG0000, B–29

control blocks, LCA0000 and CFG0000 programming notes, B–28, C–5

CPT API
CICS installation program sample, 2–36
external subroutine calls, 2–2

installation program sample, 2–35
internal subroutine calls, 2–2
task-related user exit (TRUE) interface, 2–3

custom translation table
method 1, 13–5
method 2, 13–5

customizing
RECEIVE tool, 1–6
SEND tool, 1–8
Unicenter SOLVE:CPT tools, overview, 1–3

D

data processing errors
RECEIVE tool, 1–7
SEND tool, 1–10

data SEND example, 10–6

data transfer
TCP, 2–7

data transfer, TCP programming options, 2–7

data translation
TRANSLATE service, 2–12
translating ASCII data, 2–12
translating EBCDIC data, 2–12

daughter task taking ownership of a session, 7–8

destination control table (DCT), 1–5

diagnostic code field, A–4

diagnostics
LISTEN tool, 1–5
RECEIVE tool, 1–7
SEND tool, 1–10

DSECTs. *See Assembler DSECTs*

E

EBCDIC data
translating within a user buffer, 2–12

endpoint release, using
CLOSE service, 2–11
TRUE management routines, 2–11

examples
CLOSE service
abortive close, 3–4
graceful close, 3–3
overview, 3–3
CONNECT service, 4–3
FTP client service, 5–3
GIVE service, 6–3
LISTEN service call, 7–3
LL SEND, 10–7
RCVFROM service, 8–4
RECEIVE service
double separator character example, 9–5
LL RECEIVE example, 9–4
sample daughter task taking ownership of a session, 7–8
SEND service
separator character, 10–8
SENDTO service, 11–4
TAKE service, 12–3
TRANSLATE service, 13–3, 13–4
inbound translation, 13–3
outbound translation, 13–4

exits
Unicenter SOLVE:CPT
TRUE, 5–13

EZACICAL applications, linking, D–3, D–4

EZACONFG layout, LSTP DSECT, C–3

EZASOKET applications, linking, D–2

F

facility management
GIVE service, 2–13, 2–14
multi-tasked applications, 2–13
TAKE service, 2–13, 2–15

file boundaries in TCP data transfer, 2–7

FTP Client service
arguments, 5–2
Assembler DSECTs, 5–7
complete parameter list, 5–14, 5–20
completion information, 5–8
parameter values returned in the AFT, 5–7
recommended AFT parameters, 5–2
return codes, 5–9
sample programs, 5–8
syntax, 5–2
usage example, 5–3
usage notes, 5–13

G

GIVE service
arguments for facility management (AFM), 2–14
Assembler DSECTs, 6–4
call syntax, 6–2
complete parameter list, 6–7
completion information, 6–5
completion of, 2–14
connection ownership, 6–6
defined, 6–1
example, 6–3
facility management, 2–14
invoking, 6–1
parameter values returned in the AFM, 6–4
recommended AFM parameters, 6–2
return codes, 6–5
sample programs, 6–4
token (AFMTOKEN), 6–6
usage, 6–6
version number (AFMVERS), 6–6

graceful close, 3–3

I

inbound translation example, 13–3

initialization
SEND tool transaction and data transfer, 1–8

J

JCL, D–1

L

LCA0000 and CFG0000 control blocks, programming notes, B–28, C–5

LCA0000 control block listing, B–29

linking
compile JCL, D–1
CPT/API application. *See the "Unicenter SOLVE:CPT API Services chapter.*
EZACICAL applications, D–3, D–4
EZACICSO applications, D–3
EZASOKET applications, D–2
IBM CICS Sockets (EZASOKET) application, D–3
JCL, D–1

LISTEN service, 2–6
Assembler DSECTs, 7–9
call
tokens, 7–12
complete parameter list, 7–18, 7–23
completion information, 7–11
completion of, 2–6
creating connections, 2–13
data transfer buffering, 7–16
establishing listening and client connections, 7–12
implementing TCP/IP server facilities, 7–15
initiation
with a transaction ID, 7–12
invoking, 7–1
multithreaded server
special start transaction needs, 7–5
standard, 7–4
network considerations, 7–17
parameter values returned in the ACM, 7–9
recommended ACM parameters, 7–3
recommended server, 7–4
required and optional fields, 7–15
return codes, 7–13
sample daughter task taking ownership of a session, 7–8
sample programs, 7–10
single-threaded server, 7–7
syntax, 7–2
token information, 7–15
transaction ID field (ACMTRNID), 7–15
usage notes, 7–15
version number (ACMVERS), 7–15

LISTEN service call
examples, 7–3

LISTEN tool
defining, 1–4
diagnostics, 1–5
initiating, 1–4

listen tool, invoking, 7–26

listener
invoking, 7–26

Listener
to invoke with translation from
COBOL SOLVE:CPT API, 7–27

LL RECEIVE example, 9–4

LL RECEIVE option, 9–4

LL SEND
example, 10–7
option, 10–4

LSTP DSECT, EZACONFG layout, C–3

M

macros
defining T09MSEND for customizing SEND tool, 1–8
T09MLSTN instructions, 1–6
T09MRECV instructions, 1–6
T09MTRAN, C–1

managing connections
TRUE management routines, 2–13

multi-tasked applications
facility management, 2–13

multithreaded server
special start transaction needs, 7–5
standard, 7–4

N

network considerations
LISTEN service, 7–17
RCVFROM service, 8–7
SENDTO service, 11–6

non-blocking, 9–3

non-blocking fixed length RECEIVE, 9–8

non-blocking LL RECEIVE, 9–16

non-blocking RECEIVE loop, 9–30

non-blocking separator character RECEIVE, 9–20

O

offsets
ACL control block, B–2, B–5
ADT control block
offsets, B–10
AFM control block, B–14
AXL control block, B–20
connection time security program control block, B–25

outbound translation example, 13–4

overriding the default translation table
custom, 13–5

P

parameter list passed to initiated transactions, C–2

parameter values returned in the ACL
CLOSE service:, 3–5

parameter values returned in the ACM
CONNECT service, 4–4
LISTEN service, 7–9

parameter values returned in the ADT
RCVFROM service, 8–5
RECEIVE service, 9–34
SEND service, 10–9
SENDTO service, 11–5

parameter values returned in the AFM
GIVE service, 6–4
TAKE service, 12–4

parameter values returned in the AFT
FTP lient service, 5–7

parameter values returned in the AXL
TRANSLATE service, 13–6

parsing requirements
RECEIVE tool, 1–6
writing data to the transient data queue, 1–6

processing errors
RECEIVE tool, 1–7
SEND tool, 1–9
SEND tool transport provider, 1–10

program samples
API UDP client, 2–38
API UDP server, 2–38
CICS installation, 2–35
CPT API installation, 2–32
CPT API server, 2–36, 2–37
Unicenter SOLVE
CPT API server, 2–37
Unicenter SOLVE:CPT
API CICS installation, 2–36
Unicenter SOLVE:CPT API server, 2–36

pseudo code
client condensed, 2–4
server condensed, 2–4

R

RCVFROM service
- arguments, 8–1
- complete parameter list, 8–10, 8–15
- creating UDP data transfer and endpoints, 2–10
- defined, 8–1
- network considerations, 8–7
- parameter values returned in the ADT, 8–5
- recommended ADT parameters, 8–3
- return codes, 8–8
- sample programs, 8–6
- syntax, 8–3

RECEIVE, 9–34

RECEIVE service
- return codes, 9–37

RECEIVE methodology options
- introduction, 9–6

RECEIVE service
- Assembler DSECTs, 9–34
- complete parameter list, 9–41, 9–47
- completion information, 9–36
- data
  - buffer length, 9–40
  - transfer buffering, 9–36
- double separator character example, 9–5
- invoking, 9–1
- LL RECEIVE example, 9–4
- methodology options, 9–2
- non-blocking fixed length RECEIVE, 9–8
- recommended ACM parameters
  - blocking fixed length RECEIVE, 9–24
- recommended ADT parameters, 9–8
  - blocking LL RECEIVE, 9–26
  - blocking RECEIVE loop, 9–32
  - blocking separator character RECEIVE, 9–28
  - non-blocking RECEIVE loop, 9–30
  - non-blocking variable length RECEIVE, 9–12
- recommended ADT parameters:, 9–16
- recommended ADTparameters
  - non-blocking separator character RECEIVE, 9–20
- sample program usage
  - blocking fixed length RECEIVE, 9–24
  - blocking LL RECEIVE, 9–26
  - blocking RECEIVE loop, 9–33
  - blocking separator character RECEIVE, 9–29
  - non-blocking fixed length RECEIVE, 9–9
  - non-blocking LL RECEIVE, 9–17
  - non-blocking RECEIVE loop, 9–31
  - non-blocking separator character RECEIVE, 9–21
  - non-blocking variable length RECEIVE, 9–13
- sample programs, 9–35
- syntax, 9–2
- terminology and RECEIVE concepts
  - blocking, 9–3
- token (ADTTOKEN), 9–40
- usage notes, 9–39
- version number (ADTVERS), 9–39

RECEIVE tool
- customizing, 1–6
- data processing errors, 1–7
- diagnostics, 1–7
- overview, 1–5
- parsing requirements, 1–6
- processing errors, 1–7
- reliability factors, 1–6
- transport provider processing errors, 1–7

recommended ACL parameters
- CLOSE service, 3–2

recommended ACM parameters
- CONNECT service, 4–2
- LISTEN service, 7–3
- TAKE service, 12–2

recommended ADT parameters
- RCVFROM service, 8–3
- RECEIVE service, 9–8
  - blocking fixed length RECEIVE, 9–24
  - blocking LL RECEIVE, 9–26
  - blocking RECEIVE loop, 9–32
  - blocking separator character RECEIVE, 9–28
  - non-blocking LL RECEIVE, 9–16
  - non-blocking RECEIVE loop, 9–30
  - non-blocking separator character RECEIVE, 9–20
  - non-blocking variable length RECEIVE, 9–12
- SEND service, 10–3
  - optional parameters, 10–3
- SENDTO service, 11–3

recommended AFM parameters
- GIVE service, 6–2

recommended AFT parameters
- FTP Client service, 5–2

recommended AXL parameters
- TRANSLATE service, 13–2

recommended server, LISTEN service, 7–4

RECVFROM service

Assembler DSECTs, 8–6
sample usage, 8–4

releasing connections, 2–13

reliability factors
RECEIVE tool, 1–6
SEND tool, 1–9

return codes, A–1
CONNECT service, 3–8, 4–6
diagnostic code field, A–4
FTP Client service, 5–9
GIVE service, 6–5
LISTEN service, 7–13
RCVFROM service, 8–8
RECEIVE service, 9–37
SEND service, 10–12
SENDTO service, 11–7
TAKE service, 12–6
TRANSLATE service, 13–8

S

sample DSECT member
DSCTY, B–26
T09DCSKL, B–24
T09KDACL, B–4
T09KDACM, B–8
T09KDAFM, B–16, B–18
T09KDAXL, B–23
T09KDDADT, B–12

sample program usage
RECEIVE service
blocking fixed length RECEIVE, 9–24
blocking LL RECEIVE, 9–26
blocking RECEIVE loop, 9–33
blocking separator character RECEIVE, 9–29
non-blocking fixed length RECEIVE, 9–9
non-blocking LL RECEIVE, 9–17
non-blocking RECEIVE loop, 9–31
non-blocking separator character RECEIVE, 9–21
non-blocking variable length RECEIVE, 9–13

sample programs
API UDP server, 2–38
CLOSE service, 3–6
CONNECT service, 4–5
FTP Client service, 5–8
GIVE service, 6–4
LISTEN service, 7–10
RCVFROM service, 8–6
RECEIVE service, 9–35
SEND service, 10–10
SENDTO service, 11–6
T09PAFT3, 5–5
TAKE service, 12–5
TRANSLATE service, 13–7

sample usage
RECVFROM service, 8–4

samples
T09DLST DSECT, B–27, B–28

samples
T09PCFTP, 5–3

security communications block, 2–21, 2–22

security program, 2–20

security program control block, connection time, B–25

SELECT service
complete parameter list, 10–19

SELECT tool
overview, 1–11

SELECT tool, RECEIVE service, 9–3

SEND service
arguments for ADT, 2–8
Assembler DSECTs, 10–9
call syntax, 10–2
complete parameter list, 10–15
completion information, 10–10
completion of, 2–8
data
buffer length, 10–11, 10–14
SEND example, 10–6
storage, 10–10, 10–13
defined, 10–1
invoking, 10–1
LL SEND
example, 10–7
option, 10–4
optional ADT parameters, 10–3
parameter values returned in the ADT, 10–9
queue and buffer sizes, 10–11, 10–14
recommended ADT parameters, 10–3
return codes, 10–12
sample programs, 10–10
separator character example, 10–8
separator character option, 10–4
TCP data transfer, 2–8
token (ADTTOKEN), 10–11, 10–14
version number (ADTVERS), 10–11, 10–14

SEND tool, 1–8
  control information, 1–9
  customizing, 1–8
  data processing error, 1–10
  diagnostics, 1–10
  initialization
    automated transaction and transfer data, 1–8
  processing errors, 1–9
  reliability factors, 1–9
  transport provider processing error, 1–9

SENDTO service
  Assembler DSECTs, 11–5
  call syntax, 11–3
  complete parameter list, 11–9, 11–14
  creating UDP data transfer and endpoints, 2–10
  defined, 11–1
  examples, 11–4
  network considerations, 11–6
  parameter values returned in the ADT, 11–5
  recommended ADT parameters, 11–3
  return codes, 11–7
  sample programs, 11–6

separator character RECEIVE option, 9–5

separator characters
  SEND example, 10–8
  SEND service, 10–4

server application code examples, 2–25
  multithreaded CICS, 2–27, 2–31
  multithreaded data processing, 2–29

server condensed pseudo code, 2–4

service
  complete parameter list, 11–11

setting the default translation table, 13–5

single-threaded server, 7–7

SOLVE:CPT API
  to invoke the Listener
    with translation from
      COBOL, 7–27

SOLVE:CPT FTP Client Service
  module descriptions, 5–11

SOLVE:CPT FTP Client service modules
  T09TCFCM, 5–11

SOLVE:CPT FTP Client Service modules
  T09TCFDM, 5–12
  T09TCFRM, 5–12

standard multithreaded server, 7–4, 7–5

subroutine calls
  CPT external, 2–2
  CPT internal, 2–2

syntax
  CONNECT service, 4–2
  LISTEN service, 7–2

T

T09DLST DSECT sample, B–27, B–28

T09DRTCD macro, A–4

T09DRTCD.assembler DSECT sample member, A–4

T09DSCTY DSECT member, B–25

T09MCALL Assembler macro
  body, E–1
  syntax, E–1

T09MCALL macro
  overview, E–1
  service keywords, E–1
  usage example, E–2

T09MLSTN
  macro, C–1

T09MLSTN macro
  instructions, 1–6

T09MRECV macro
  instructions, 1–6

T09MSEND macro
  defining for customizing SEND tool, 1–8

T09MTRAN
  programming notes, C–1

T09MTRAN initiated transactions
  parameter list passed, C–2

T09MTRAN statement
  basic, recommended layout for data passed from PARM field, C–2
  EZACONFG layout, C–3

T09PAFT3, sample program, 5–5

T09PCFTP
  sample program, 5–3

T09TCFCM
  SOLVE:CPT FTP Client service module, 5–11

T09TCFDM

SOLVE:CPT FTP Client Service module, 5–12

T09TCFRM
SOLVE:CPT FTP Client Service module, 5–12

T09TRAN
programming notes, E–1

TAKE service
Assembler DSECTs, 12–4
complete parameter list, 12–9
completion information, 12–5
completion of, 2–15
data transfer, 12–7
defined, 12–1
examples, 12–3
facility management, 2–15
implementing, 2–15
overview, 12–1
parameter values returned in the AFM, 12–4
recommended ACM parameters, 12–2
return codes, 12–6
sample programs, 12–5
token (AFMTOKEN), 12–8
usage notes, 12–7
version number (AFMVERS), 12–8

task-related user exit (TRUE) interface, 2–3

TCP
connection management, 2–5
argument for connection management, 2–5
token connection, 2–5
data transfer, 2–7
client/server mechanisms, 2–7
file boundaries, 2–7
programming options, 2–7
RECEIVE service, 2–8
SEND service, 2–8

terminology and RECEIVE concepts
blocking, 9–3
LL RECEIVE option, 9–4
non-blocking, 9–3
SELECT tool, 9–3
separator
RECEIVE option, 9–5

token connection, 2–5
CONNECT service, 2–6
LISTEN service, 2–6

token UDP endpoints, 2–9

TRANSLATE service
arguments for translation (AXL), 2–12
Assembler DSECTs, 13–6
call syntax, 13–2
complete parameter list, 13–10, 13–11
completion information, 13–7
completion of, 2–12
defined, 13–1
inbound translation example, 13–3
invoking, 13–1
outbound translation example, 13–4
overriding the default translation table
custom, 13–5
parameter values returned in the AXL, 13–6
recommended AXL parameters, 13–2
return codes, 13–8
sample programs, 13–7
setting the default translation table, 13–5
token (AXLTOKEN), 13–9
usage notes, 13–9
version number (AXLVERS), 13–9

translation table
custom
method 2, 13–5

transport provider errors
RECEIVE tool, 1–7
SEND tool, 1–9

TRUE interface, 2–3

TRUE management routines
connection and endpoint release, 2–11
managing connections, 2–13

U

UDP data transfer and endpoints
creating
RCVFROM service, 2–9
SENDTO service, 2–9

Unicenter SOLVE:CPT
Administrative Interface
overview, 1–2
API
application programming concepts, 2–3
CICS installation program sample, 2–32
invoking listener tool, 7–26
server program samples, 2–37
services, 1–2
UDP client programs sample, 2–38
UDP server programs sample, 2–38
well-known transport provider port, 2–3
FTP Client service
callable service, 1–2

diagram, 2–18
introduction, 2–16
overview, 2–17
interface, 2–2
tools
CICS code, 1–3
customizing, 1–3
LISTEN, 1–4
RECEIVE, 1–5, 1–6
SEND, 1–8
TRUE exits, 5–13

Unicenter SOLVE:CPT TRUE exits, 5–13

Unicenter TCPaccess
connection management
CONNECT service, 2–6
LISTEN service, 2–6

usage examples
FTP client service, 5–3

usage notes
CLOSE service, 3–9
CONNECT service, 4–8
FTP Client service, 5–13
GIVE service, 6–6
LISTEN service, 7–15
RECEIVE service, 9–39
SEND service, 10–13
TAKE service, 12–7
TRANSLATE service, 13–9

W

well-known port, 2–3

X

xTRANSLATE service:, 13–3