OpenTP1 Version 7

TP1/Client User's Guide
TP1/Client/W, TP1/Client/P

3000-3-D58-20(E)

■ Relevant program products

Note: In the program products listed below, those marked with an asterisk (*) might be released later than the other program products.

For AIX 5L V5.1, AIX 5L V5.2, AIX 5L V5.3, and AIX 6.1

P-1M64-2531 uCosminexus TP1/Client/W 07-02

For HP-UX 11i and HP-UX 11i V2 (PA-RISC)

R-18451-41K uCosminexus TP1/Client/W 07-02*

For HP-UX 11i V2 and HP-UX 11i V3 (IPF)

R-18451-21J uCosminexus TP1/Client/W 07-02*

For Solaris 8, Solaris 9, and Solaris 10

R-19451-216 uCosminexus TP1/Client/W 07-02*

For Red Hat Enterprise Linux 5.1 Advanced Platform and Red Hat Enterprise Linux 5.1 (x86, AMD64, or Intel EM64T)

P-9S64-2561 uCosminexus TP1/Client/W 07-02*

For Red Hat Enterprise Linux 5.1 Advanced Platform and Red Hat Enterprise Linux 5.1 (IPF)

P-9V64-2531 uCosminexus TP1/Client/W 07-02*

For Windows Vista, Windows Server 2003 x64 Editions, Windows Server 2003, Windows XP, and Windows 2000 (32-bit edition)

P-2464-2144 uCosminexus TP1/Client/P 07-02*

This manual can be used for other products, in addition to the products shown above. For details, see the *Release Notes*.

This product has been developed in accordance with a quality system approved under ISO 9001 and TickIT.

■ Edition history

Edition 1 (3000-3-D58(E)): June 2006

Edition 2 (3000-3-D58-20(E)): March 2009

■ Copyright

Summary of amendments

The following table lists changes in this manual (3000-3-D58-20(E)) and product changes related to this manual for uCosminexus TP1/Client/W 07-02 and uCosminexus TP1/Client/P 07-02.

Changes	Location
A facility that assembles messages and a facility for checking whether a message has been delivered have been added. With this addition, the following functions have been added: • `dc_clt_assem_send_s` • `dc_clt_assem_receive_s` • `CBLDCCLS('ASMSEND')` • `CBLDCCLS('ASMRECV')` The following operand has been added to the client environment definition: • `DCCLTDELIVERYCHECK` The description of the following message has been changed: • KFCA02447-E The following messages have been added: • KFCA02485-E • KFCA02486-E	*2.5.1*, *2.5.2*, *2.5.3*, *2.5.4*, *2.5.5*, *3.1.1*, *4.6.4*, *4.6.5*, *5.1.1*, *6.6.5*, *6.6.6*, *7.2.1*, *7.2.2*, *7.2.5*, *8.2.1(4)*, *8.2.1(5)*, and *10.3*
Identification information for performance verification set by TP1/Client can now be included in the TP1/Server performance verification trace. With this change, the `DCCLTPRFINFOSEND` operand has been added to the client environment definition.	*2.11.5*, *7.2.1*, *7.2.2*, and *7.2.5*
A facility that specifies the sending host for a TP1/Client CUP has been added. With this addition, the `DCCLTCUPSNDHOST` operand has been added to the client environment definition.	*2.12.3*, *2.13*, *7.2.1*, *7.2.2*, and *7.2.5*
An explanation of the facility that fixes the receive port has been added.	*2.14*
Notes on using functions and request statements have been added.	*4.1* and *6.1*
The explanation of the file loading order used when a path name is specified in the `defpath` argument has been changed for the following functions: • `dc_clt_cltin_s` function • `dc_clt_accept_notification_s` function • `dc_clt_cancel_notification_s` function • `dc_clt_open_notification_s` function	*4.2.1(3)*, *4.7.1(3)*, *4.7.2(3)*, and *4.7.3(3)*

Changes	Location
The explanation of the file loading order used when a path name is specified as a data name has been changed for the following request statements: • `CBLDCCLS('CLTIN   ')` • `CBLDCCLS('EXCLTIN ')` • `CBLDCCLS('NOTIFY  ')` • `CBLDCCLS('EXNACPT ')` • `CBLDCCLS('CANCEL  ')` • `CBLDCCLS('EXNCANCL')` • `CBLDCCLS('O-NOTIFY')`	*6.2.1(3)*, *6.2.2(3)*, *6.7.1(3)*, *6.7.2(3)*, *6.7.3(3)*, *6.7.4(3)*, and *6.7.5(3)*
The explanation of the library name to be specified as a linkage option has been changed.	*5.2.2*
data-name-C has been added to the explanation of data areas to which `CBLDCRPS('GETWATCH')` returns values.	*6.3.5(4)*
`CBLDCCLS('STCONIF ')` has been added as a request statement for setting terminal identification information.	*6.4.5*
The notes on the following functions have been changed: • `dc_clt_send_s` function • `CBLDCCLS('SEND    ')` • `CBLDCCLS('EXSEND  ')`	*4.6.1(5)*, *6.6.1(6)*, and *6.6.2(6)*
An explanation that applies when a value from 1 to 65535 is set has been added for *data-name-E* of `CBLDCCLS('RECEIVE2')`.	*6.6.4(3)*
An explanation that request statements provided by the character code converter can operate correctly in a multi-thread environment has been added.	*6.8* and *6.9*
A list of client environment definition items has been added.	*7.1*
The range of specifiable values for the `DCSCDMULTICOUNT` operand in the client environment definition has been changed to 1 to 4096, and the default value for the operand has been changed to 1.	*7.2.5*
Version changes for the following versions have been added: • TP1/Client/W 07-02 • TP1/Client/P 07-02	*B.1*

The following table lists changes in this manual (3000-3-D58-20(E)) and product changes related to this manual for uCosminexus TP1/Client/W 07-01 and uCosminexus TP1/Client/P 07-01.

Changes	Location
The `dc_clt_receive2_s` function has been added to the functions that can be executed from a CUP to receive messages from an MHP.	*2.5.3*
An explanation of executing functions that are not suitable for a multi-thread environment has been added.	*2.9.2* and *2.9.3*
XATMI interface functions are no longer able to operate in a multi-thread environment.	*3.1.1*, *4*, and *4.8*
For the `dc_clt_cltin_s` function, how the following arguments are specified has been changed: • `logname` • `passwd`	*4.2.1(3)*
An explanation that a client environment definition can be specified each time for each function call has been added for the following functions: • `dc_clt_cltin_s` function • `dc_clt_accept_notification_s` function • `dc_clt_cancel_notification_s` function • `dc_clt_open_notification_s` function	*4.2.1*, *4.7.1*, *4.7.2*, *4.7.3*, *7.2.8*, and *7.2.9*
The directory of COBOL templates for TP1/Client/W has been changed.	*5.3*
The following operands have been added to the client environment definition to prevent data transmission delays between TP1/Client and TP1/Server Base: • `DCCLTRECVBUFSIZE` • `DCCLTSENDBUFSIZE` • `DCCLTTCPNODELAY`	*7.2.1*, *7.2.2*, and *7.2.5*
If the name of the service group called by the `dc_rpc_call_s` function has not been defined in the file specified in the `DCCLTSERVICEGROUPLIST` operand, the `DCCLTNOSERVER` operand in the client environment definition determines how TP1/Client operates. An explanation to this effect has been added for the `DCCLTSERVICEGROUPLIST` operand.	*7.2.5*

Changes	Location
An explanation stating the following has been added: If TP1/Client connects to a RAP-processing server, the operands in the RAP-processing listener service definition take precedence for the following operands: • `DCCLTTREXPTM` • `DCCLTTREXPSP` • `DCCLTTRCPUTM` • `DCCLTINQUIRETIME` • `DCCLTTRSTATISITEM` • `DCCLTTROPTIITEM` • `DCCLTTRWATCHTIME` • `DCCLTTRRBINFO` • `DCCLTTRLIMITTIME` • `DCCLTTRRBRCV` • `DCCLTTRRECOVERYTYPE`	*7.2.5*
An explanation stating that the following functions read definitions from the file specified in the `defpath` argument has been added: • `dc_clt_cltin_s` function • `dc_clt_accept_notification_s` function • `dc_clt_cancel_notification_s` function • `dc_clt_open_notification_s` function	*7.2.8* and *7.2.9*
Messages have been added.	*KFCA02445-E*, *KFCA02446-E*, *KFCA02447-E*, *KFCA02450-W*, and *KFCA02451-W*
An explanation of changes made to functions, definitions, and messages when the TP1/Client version was updated has been added.	*Appendix B*

In addition to the above changes, minor editorial corrections have been made.

Preface

This manual describes the functionality and use of the program products listed below:

- P-1M64-2532 uCosminexus TP1/Client/W 07-00
- P-2464-2147 uCosminexus TP1/Client/P 07-00

Products described in this manual, other than those for which the manual is released, may not work with OpenTP1 Version 7 products. You need to confirm that the products you want to use work with OpenTP1 products.

Intended readers

This manual is intended for system managers, system designers, programmers, and operators.

Readers should first look at the manual *OpenTP1 Description* which introduces OpenTP1.

Organization of this manual

This manual is organized as follows:

1. Overview

Outlines OpenTP1 Client features.

2. Facilities

Describes OpenTP1 client facilities.

3. User Application Program Interface (C Language)

Describes the user application program interface in C.

4. TP1/Client Functions (C Language)

Describes the functions that can be used in TP1/Client.

5. User Application Program Interface (COBOL Language)

Describes the user application program interface in COBOL.

6. Request Statements Available for TP1/Client (COBOL Language)

Describes the request statements that can be used in TP1/Client.

7. Definition

Explains the client environment definition.

8. Operating Commands

Explains how to enter, code, and use TP1/Client operating commands.

9. Error Recovery

Explains how to deal with errors.

10. Messages

Explains TP1/Client messages.

A. Code Conversion Specifications

Explains the specifications of the code conversion performed by the character code converter.

B. Version Changes

Explains the changes to functions, definitions, and messages made when the TP1/Client version was updated.

Related publications

This manual is part of a related set of manuals. The manuals in the set, including this manual, are listed below. The manual numbers follow the manual titles.

OpenTP1 products

- *OpenTP1 Version 7 Description* (3000-3-D50(E))
- *OpenTP1 Version 7 Programming Guide* (3000-3-D51(E))
- *OpenTP1 Version 7 System Definition* (3000-3-D52(E))
- *OpenTP1 Version 7 Operation* (3000-3-D53(E))
- *OpenTP1 Version 7 Programming Reference C Language* (3000-3-D54(E))
- *OpenTP1 Version 7 Programming Reference COBOL Language* (3000-3-D55(E))
- *OpenTP1 Version 7 Messages* (3000-3-D56(E))
- *OpenTP1 Version 7 Tester and UAP Trace User's Guide* (3000-3-D57(E))
- *OpenTP1 Version 7 TP1/Client User's Guide TP1/Client/W, TP1/Client/P* (3000-3-D58(E))
- *OpenTP1 Version 7 TP1/Client User's Guide TP1/Client/J* (3000-3-D59(E))
- *OpenTP1 Version 7 Protocol TP1/NET/TCP/IP* (3000-3-D70(E))

Other OpenTP1 products

- *TP1/Web User's Guide and Reference* (3000-3-D62(E))[1]

Other related products

- *VOS3 Data Management System XDM E2 Description (6190-6-620(E))*
- *VOS3 Data Management System XDM E2 System Definition (6190-6-625(E))*

Note

You must check and confirm that the products described in these manuals will run on your operating system.

[1] If you want to use this manual, confirm that it has been published. (Some of these manuals might not have been published yet.)

Conventions: Abbreviations

This manual uses the following abbreviations for product names:

Full name	Abbreviation
AIX 5L V5.1	AIX
AIX 5L V5.2	
AIX 5L V5.3	
AIX 6.1	
HP-UX 11i (PA-RISC)	HP-UX
HP-UX 11i V2 (IPF)	
HP-UX 11i V2 (PA-RISC)	
HP-UX 11i V3 (IPF)	
Itanium(R) Processor Family	IPF
Red Hat Enterprise Linux 5.1 (AMD64)	Linux
Red Hat Enterprise Linux 5.1 (Intel EM64T)	
Red Hat Enterprise Linux 5.1 (IPF)	
Red Hat Enterprise Linux 5.1 (x86)	
Red Hat Enterprise Linux 5.1 Advanced Platform	
Microsoft$^{(R)}$ MS-DOS$^{(R)}$	MS-DOS
Solaris 8	Solaris
Solaris 9	
Solaris 10	

Full name	Abbreviation	
uCosminexus TP1/Client/P	TP1/Client/P	TP1/Client
uCosminexus TP1/Client/W	TP1/Client/W	
uCosminexus TP1/Extension 1	TP1/Extension 1	
uCosminexus TP1/NET/TCP/IP	TP1/NET/TCP/IP	
uCosminexus TP1/Online Tester	TP1/Online Tester	
uCosminexus TP1/LiNK	TP1/LiNK	TP1/Server
uCosminexus TP1/Server Base	TP1/Server Base	
Microsoft$^{(R)}$ Windows$^{(R)}$ 2000 Advanced Server Operating System	Windows 2000	
Microsoft$^{(R)}$ Windows$^{(R)}$ 2000 Datacenter Server Operating System		
Microsoft$^{(R)}$ Windows$^{(R)}$ 2000 Professional Operating System		
Microsoft$^{(R)}$ Windows$^{(R)}$ 2000 Server Operating System		
Microsoft$^{(R)}$ Windows$^{(R)}$ Software Development Kit	Windows SDK	
Microsoft$^{(R)}$ Windows Server$^{(R)}$ 2003, Datacenter Edition	Windows Server 2003	
Microsoft$^{(R)}$ Windows Server$^{(R)}$ 2003, Datacenter x64 Edition		
Microsoft$^{(R)}$ Windows Server$^{(R)}$ 2003, Enterprise Edition		
Microsoft$^{(R)}$ Windows Server$^{(R)}$ 2003, Enterprise x64 Edition		
Microsoft$^{(R)}$ Windows Server$^{(R)}$ 2003 R2, Enterprise Edition		
Microsoft$^{(R)}$ Windows Server$^{(R)}$ 2003 R2, Enterprise x64 Edition		
Microsoft$^{(R)}$ Windows Server$^{(R)}$ 2003 R2, Standard Edition		
Microsoft$^{(R)}$ Windows Server$^{(R)}$ 2003 R2, Standard x64 Edition		
Microsoft$^{(R)}$ Windows Server$^{(R)}$ 2003, Standard Edition		
Microsoft$^{(R)}$ Windows Server$^{(R)}$ 2003, Standard x64 Edition		
Microsoft$^{(R)}$ Windows Vista$^{(R)}$ Business (x86)	Windows Vista (32bit)	Windows Vista
Microsoft$^{(R)}$ Windows Vista$^{(R)}$ Enterprise (x86)		

Full name	Abbreviation	
Microsoft$^{(R)}$ Windows Vista$^{(R)}$ Ultimate (x86)		
Microsoft$^{(R)}$ Windows Vista$^{(R)}$ Business (x64)	Windows Vista (64bit)	
Microsoft$^{(R)}$ Windows Vista$^{(R)}$ Enterprise (x64)		
Microsoft$^{(R)}$ Windows Vista$^{(R)}$ Ultimate (x64)		
Microsoft$^{(R)}$ Windows$^{(R)}$ XP Home Edition Operating System	Windows XP	
Microsoft$^{(R)}$ Windows$^{(R)}$ XP Professional Operating System		

- In this manual, Windows 2000, Windows Server 2003, Windows XP, and Windows Vista are generally referred to as Windows.
- In this manual, AIX, HP-UX, Linux, and Solaris are generally referred to as UNIX.

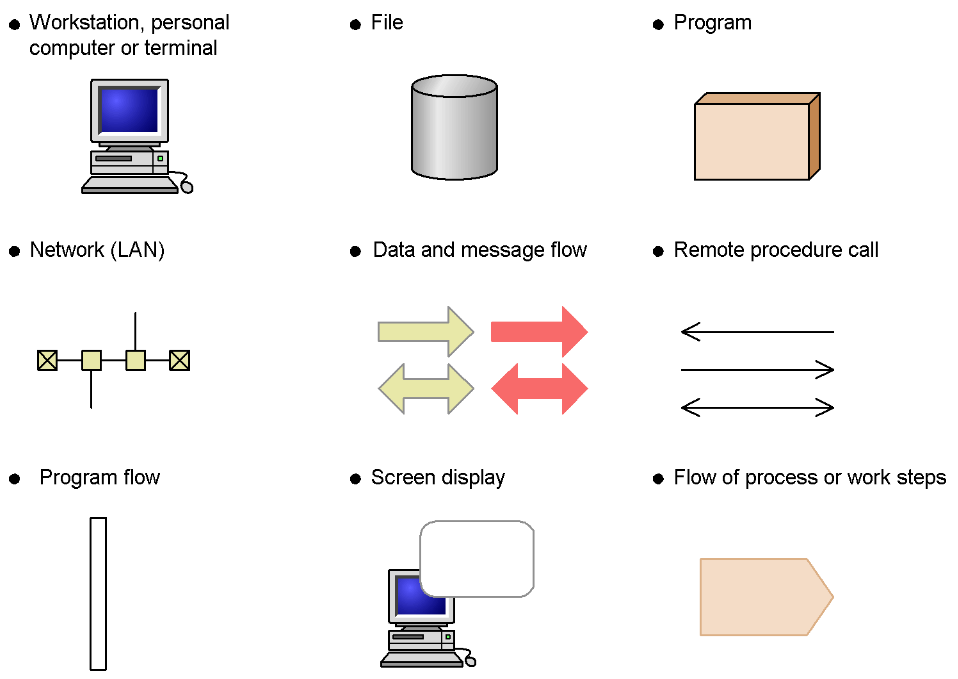
Abbreviations used in this manual are listed below.

Abbreviation	Full name
API	Application Programming Interface
CUP	Client User Program
DAM	Direct Access Method
DLL	Dynamic Linking Library
EBCDIC	Extended Binary Coded Decimal Interchange Code
ID	Identifier
JIS	Japanese Industrial Standard
KEIS	Kanji Processing Extended Information System Code
MHP	Message Handling Program
MTU	Maximum Transmission Unit
OLTP	Online Transaction Processing
PC	Personal Computer
PRF	PeRFormance
RPC	Remote Procedure Call

Abbreviation	Full name
SPP	Service Providing Program
TCP/IP	Transmission Control Protocol/Internet Protocol
UAP	User Application Program
WAN	Wide Area Network
WS	Workstation

Conventions: Diagrams

The components of the diagrams in this manual have the following meanings unless otherwise specified:



Conventions: KB, MB, GB, and TB

This manual uses the following conventions:

- 1 KB (kilobyte) is 1,024 bytes.
- 1 MB (megabyte) is 1,024^2 bytes.
- 1 GB (gigabyte) is 1,024^3 bytes.

- 1 TB (terabyte) is $1{,}024^4$ bytes.

Conventions: Platform-specific notational differences

For the Windows version of OpenTP1, there are some notational differences from the description in the manual. The following table describes these differences.

Item	Description in the manual	Change to:
Environment variable	*$aaaaaa* Example: `$DCDIR`	*%aaaaaa%* Example: `%DCDIR%`
Path name separator	Colon (`:`)	Semicolon (`;`)
Directory name separator	Slash (`/`)	Backslash (`\`)
Absolute path name	A path from the root directory Example: `/tmp`	A path name from a drive letter and the root directory Example: `C:\tmp`
Executable file name	File name only (without an extension) Example: `mcfmngrd`	File name with an extension Example: `mcfmngrd.exe`
make command	`make`	`nmake`

Conventions: Version numbers

Note that the version numbers of Hitachi program products are often written as two sets of two digits separated by a hyphen. For example:

- version 1.00 (or version 1.0) is written as 01-00
- version 2.05 is written as 02-05
- version 2.50 (or version 2.5) is written as 02-50
- version 12.25 is written as 12-25

So, for example, the version number written on the spine of a manual might take the form *Ver. 2.00* but the version number written in the program might take the form *02-00*. These numbers indicate the same version.

Acknowledgements

Quotations from X/Open CAE Specification Distributed Transaction Processing: The XATMI Specification published by X/Open Company Limited

The following section comes from Chapter 5 COBOL Reference Manual Pages of the above document.

Chapter 4. User Application Program Interface (C Language)

4.8 XATMI interface facility (tp~)

COBOL acknowledgments

COBOL was developed by CODASYL (the Conference on Data Systems Languages). The publisher of this manual expresses acknowledgment to the original developer and presents the following acknowledgment statement as requested by CODASYL. This statement is quoted from the acknowledgment in the original CODASYL COBOL specification titled *COBOL Journal of Development* 1984.

Any organization interested in reproducing the COBOL report and specifications in whole or in part, using ideas from this report as the basis for an instruction manual or for any other purpose, is free to do so. However, all such organizations are requested to reproduce the following acknowledgment paragraphs in their entirety as part of the preface to any such publication. Any organization using a short passage from this document, such as in a book review, is requested to mention "COBOL" in acknowledgment of the source, but need not quote the acknowledgment.

COBOL is an industry language and is not the property of any company or group of companies, or of any organization or group of organizations.

No warranty, expressed or implied, is made by any contributor or by the CODASYL COBOL Committee as to the accuracy and functioning of the programming system and language. Moreover, no responsibility is assumed by any contributor, or by the committee, in connection therewith.

The authors and copyright holders of the copyrighted material used herein

FLOW-MATIC (trademark of Sperry Rand Corporation), Programming for the Univac (R) I and II, Data Automation Systems copyrighted 1958, 1959, by Sperry Rand Corporation; IBM Commercial Translator From No. F28-8013, copyrighted 1959 by IBM; FACT, DSI 27A5260-2760, copyrighted 1960 by Minneapolis-Honeywell

have specifically authorized the use of this material in whole or in part, in the COBOL specifications. Such authorization extends to the reproduction and use of COBOL specifications in programming manuals or similar publications.

Important note

Please check the availability of the products and manuals for HAmonitor, ServerConductor/DeploymentManager, Cosminexus, and Job Management Partner 1/ Automatic Job Management System 2.

Contents

List of figures

List of tables

Chapter

1. Overview

This chapter contains the following sections:

1.1 TP1/Client features

OpenTP1 Client (TP1/Client) allows a workstation (WS) or personal computer (PC) to request services from an OpenTP1 server UAP using remote procedure calls. The WS or PC program that requests a service is called a CUP. The server UAP is called an OpenTP1 service providing program (SPP).

WS: Workstation

PC: Personal Computer

CUP: Client User Program

SPP: Service Providing Program

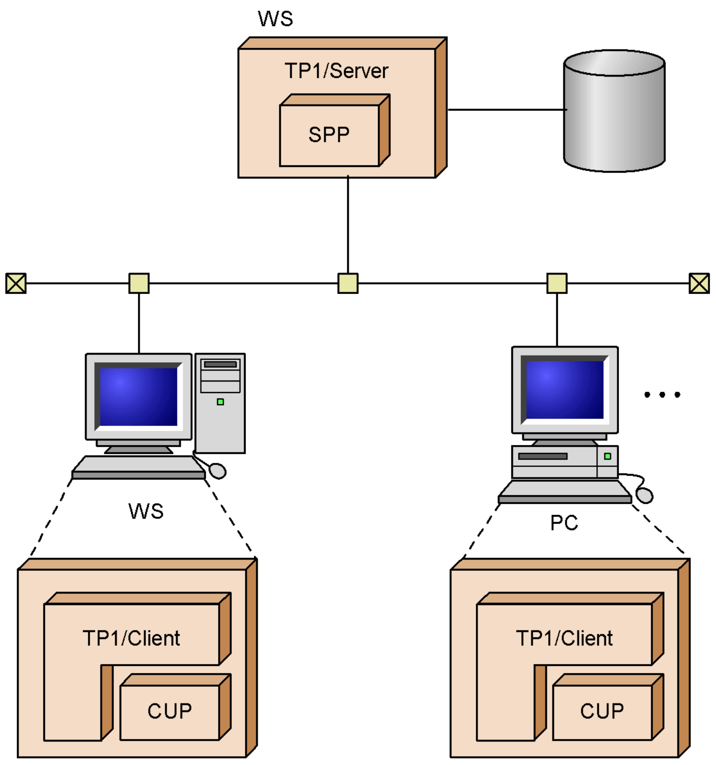
By using TP1/Client, transactions can be started from a SPP activated by a CUP. This enables construction of a distributed OLTP environment using LAN-connected WSs or PCs as clients.

This manual refers to TP1/Client/W and TP1/Client/P when describing the WS and PC versions of TP1/Client, respectively.

The manual also uses TP1/Server as a generic name for TP1/Server Base and TP1/LiNK.

The following figure shows the relationship between TP1/Server and the two TP1/Client versions.

Figure 1-1: Relationship between TP1/Server and TP1/Client



1.2 TP1/Client operation

When executing the following facilities, TP1/Client uses a specific TP1/Server as a gateway.

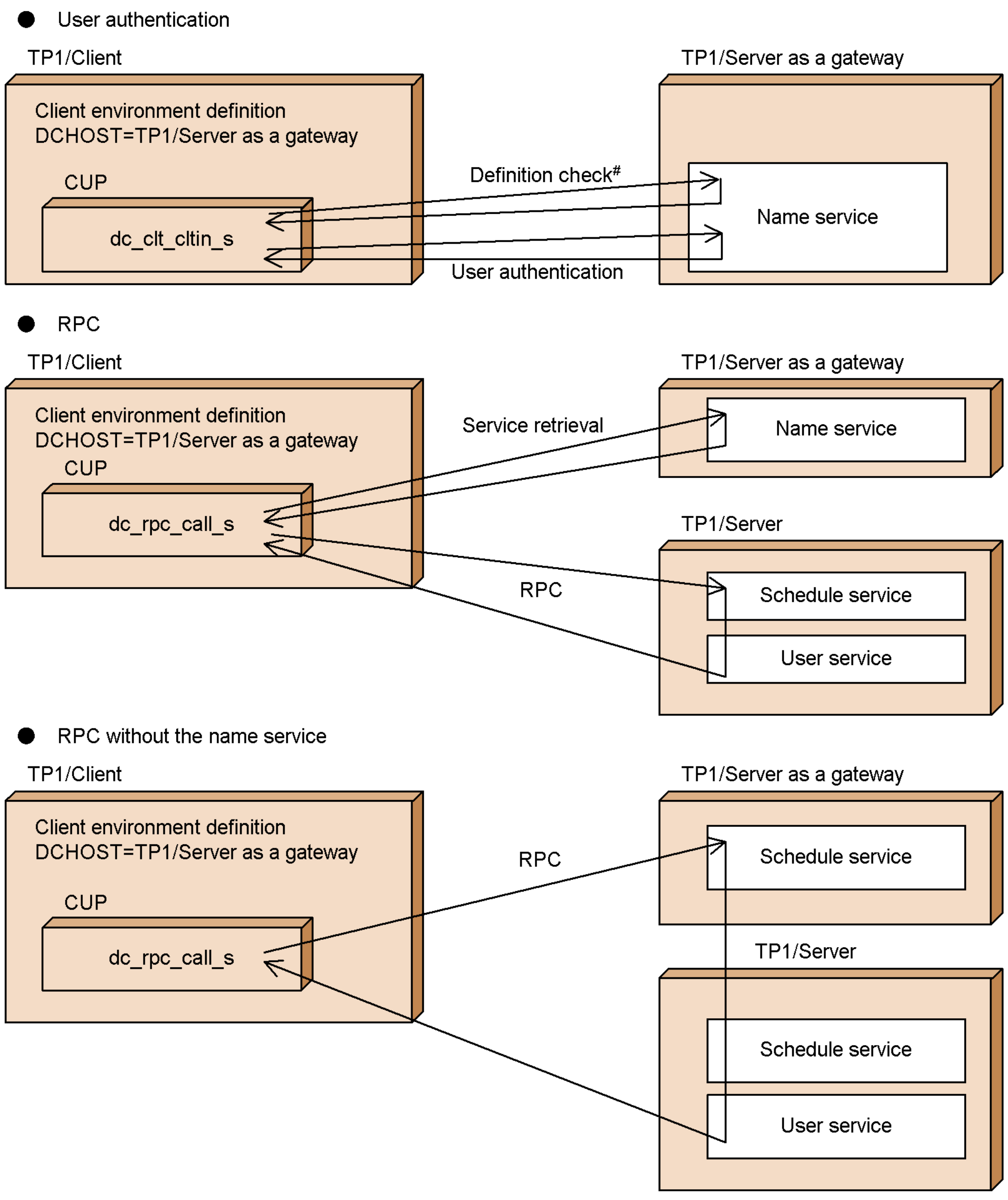
- User authentication facility
- Remote procedure call (RPC)
- RPC without the name service
- RPC for allocating a process that manages transactions

  RPC: Remote Procedure Call

Specify the gateway TP1/Server using `DCHOST` in the client environment definition or using a function argument when requesting user authentication.

The following figures show how TP1/Client operates.

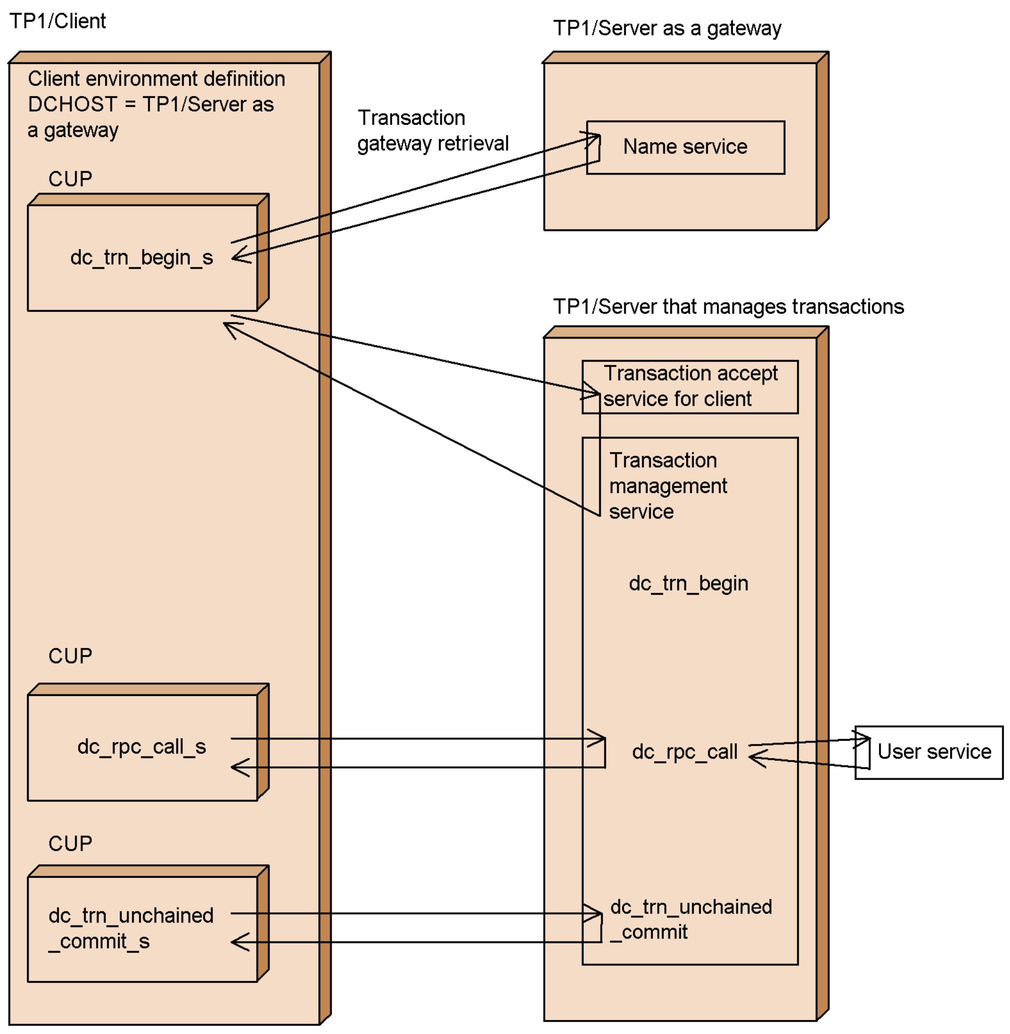
Figure 1-2: TP1/Client operation (1/2)



#: The client_uid_check operand specification in the system common definition is checked.

Figure 1-3: TP1/Client operation (2/2)

- RPC for allocating a process that manages transactions


TP1/Client
Client environment definition
DCHOST = TP1/Server as a gateway
CUP
dc_trn_begin_s
CUP
dc_rpc_call_s
CUP
dc_trn_unchained _commit_s
Transaction gateway retrieval
TP1/Server as a gateway
Name service
TP1/Server that manages transactions
Transaction accept service for client
Transaction management service
dc_trn_begin
dc_rpc_call
dc_trn_unchained _commit
User service

Chapter 2. Facilities

This chapter describes the client facilities of OpenTP1.

In this chapter, C functions (`dc_`*xxx*`_`*xxx*`_s`) are used to call the DLL for a facility. If you use functions of the normal object library (`dc_`*xxx*`_`*xxx*) or COBOL, replace the C function names with the corresponding function or facility names.

This chapter contains the following sections:

2.1 User authentication

User authentication is a TP1/Client facility for restricting the client users that receive services from TP1/Server. To receive TP1/Server services, the user must be registered at the TP1/Server host.

User authentication can be used only when the version of TP1/Server Base is 01-02 or later.

2.1.1 Implementing user authentication

Before requesting a service from TP1/Client to TP1/Server, you need to let the CUP execute the `dc_clt_cltin_s` function. Specify the user name and password to authenticate the user for receiving services from TP1/Server.

This server authenticates the client user based on the OpenTP1 security server or user management information for UNIX, Windows NT or Windows 2000.

To use user authentication, preparation is required on the server side. The preparation differs depending on whether the security server is used.

When the security server is used for authentication:

> Register the user name, group name, and password in the OpenTP1 registry.

When you want to use the same authentication as for a login to the system where TP1/Server is running without using the security server:

> Client user authentication is performed based on the user management information. In a UNIX system, register the login name and password in `/etc/passwd`. In a Windows NT 4.0 system or Windows 2000 system, use User Manager to register the users.

Note that when `client_uid_check=N` is specified in the system common definition of TP1/Server, the users not registered in the user management information can also be authenticated.

In a Windows environment, more than one CUP can run concurrently. Therefore, user authentication is required for each CUP. To perform user authentication for each CUP, TP1/Client provides API functions for user authentication. Before a CUP executes the `dc_rpc_open_s` function, the CUP executes the `dc_clt_cltin_s` function to be permitted to execute `dc_rpc_open_s`.

2.1.2 Specifying TP1/Server for authentication request

Evaluate and determine TP1/Server for which you request authentication in the following preferences.

1. Node specified for an argument of the `dc_clt_cltin_s` function

2. Node specified for `DCHOST` in the client environment definition
3. Node that first returns a response to the inquiry issued to TP1/Server

2.1.3 Suppressing user authentication

You may want to suppress user authentication (prevent communication from occurring) when you use the remote API facility, for example. To suppress user authentication, specify `DCCLTAUTHENT=N` in the client environment definition or specify `DCCLT_NO_AUTHENT` in the `flags` argument of the `dc_clt_cltin_s` function.

You must issue the `dc_clt_cltin_s` function even when suppressing user authentication.

2.1.4 Communicating with a server other than TP1/Server

In an environment where TP1/Server does not exist, you can communicate with a DCCM3 logical terminal or another server by specifying `DCCLTNOSERVER=Y` in the client environment definition.

You must issue the `dc_clt_cltin_s` function even when communicating with a server other than TP1/Server. In the `logname` argument of the `dc_clt_cltin_s` function, specify a value other than NULL. If you specify NULL, the `dc_clt_cltin_s` function will return with the `DCCLTER_INVALID_ARGS` error.

2.2 Permanent connection

TP1/Client allows the message exchange by maintaining connection between a CUP and the server. This connection method is called *permanent connection*. You can specify logical terminals for TP1/Server and VOS3 XDM/DCCM3 as remote servers for which you request to establish a permanent connection. The permanent connection decreases control packets for establishing and releasing connection, providing effective communication.

For brevity, this manual calls *VOS3 XDM/DCCM3* simply *DCCM3*.

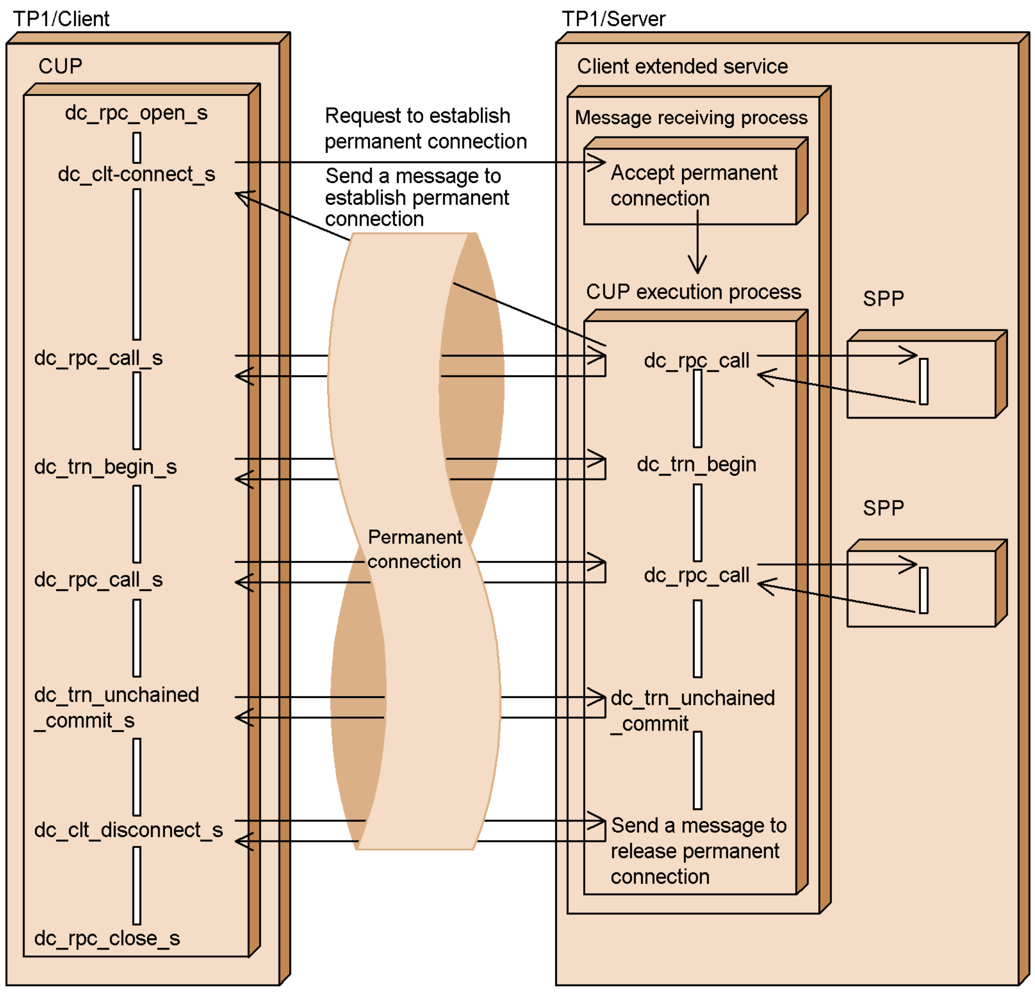
2.2.1 Establishing and releasing the permanent connection

Using the `dc_clt_connect_s` function, a CUP requests the server's client extended service to establish permanent connection. When the process accepts an establishment request, it passes that request to the CUP execution process. When the CUP receives an establishment acknowledgment message from the CUP execution process, permanent connection is established between the CUP and the CUP execution process.

A one-to-one correspondence is established between a CUP and a CUP execution process. One OpenTP1 node can simultaneously accept as many CUPs as the maximum number of processes for the CUP execution process (the specification of `cup_parallel_count` in the client service definition).

Then the succeeding messages are transferred through the permanent connection until the CUP releases it using the `dc_clt_disconnect_s` function. An error also releases the permanent connection. The following figure shows how permanent connection is established and released.

Figure 2-1: Establishing and releasing permanent connection



2.2.2 Definitions needed for permanent connection

When you use permanent connection, you need the following definitions.

- Client environment definition

  `DCCLTINQUIRETIME`

  `DCCLTPORT`

  `DCCLTDCCMHOST`

  `DCCLTDCCMPORT`

- Client service definition

  `clt_inquire_time`

  `clt_port`

  `clt_cup_conf`

  `cup_parallel_count`

  `cup_balance_count`

2.2.3 Reporting terminal identification information to the DCCM3 logical terminal

When you use a permanent connection to communicate with a DCCM3 logical terminal, you can report the terminal identification information that you specify on the client (TP1/Client) to the DCCM3 logical terminal. Therefore, you can use the DCCM3's function for allocating a fixed terminal.

(1) Exchanging messages at the DCCM3 logical terminal

A DDCM3 logical terminal exchanges messages with another logical terminal. When a DCCM3 logical terminal communicates with a CUP of TP1/Client, the DCCM3 logical terminal uses the IP address and the DCCM3 logical terminal's port number to identify the CUP as a logical terminal.

When a DCCM3 logical terminal communicates with a multi-thread CPU, a service request from any thread has the same IP address and port number. Therefore, if several DCCM3 logical terminals that communicate with a multi-thread CUP are defined on the DCCM3 side, the combination of the CUP and the DCCM3 logical terminal is not unique. This may cause problems to some applications since the sequence of DCCM3 server processing is not guaranteed when different DCCM3 logical terminals receive service requests.

(2) Reporting terminal identification information

When you use a permanent connection to communicate with a DCCM3 logical terminal, you can report the terminal identification information that you specify on TP1/Client to the DCCM3 logical terminal. Therefore, you can use the DCCM3's function for allocating a fixed terminal. Use the logical terminal name of the DCCM3 logical terminal as the terminal identification information.

By reporting the terminal identification information to the DCCM3 logical terminal, the CUP is always allocated to the same logical terminal. Therefore, you can identify the combination of the CUP and the logical terminal.

To report the terminal identification information to the DCCM3 logical terminal, specify the terminal identification information in `DCCLTCONNECTINF` of the client environment definition. Alternatively, execute the `dc_clt_set_connect_inf_s` function to set the terminal identification information.

When you use a permanent connection to communicate with the DCCM3 logical terminal, the following two methods are available. However, only method 2 can report terminal identification information to the DCCM3 logical terminal.

1. Specify the host name of the DCCM3 logical terminal in the `DCCLTDCCMHOST` client environment definition and the port number of the DCCM3 logical terminal in `DCCLTDCCMPORT`. Specify `DCCLT_DCCM3` in the flags argument of the `dc_clt_connect_s` function.
2. Specify the host name and the port number of the DCCM3 logical terminal in the `DCCLTRAPHOST` client environment definition and specify `DCNOFLAGS` in the flags argument of the `dc_clt_connect_s` function.

(3) Notes on reporting terminal identification information to the DCCM3 logical terminal

- Reporting terminal identification information enables you to use DCCM3's function for allocating a fixed terminal only when you use DCCM3 version 09-03 and later. For details about the function for allocating a fixed terminal, see the manual *VOS3 Data Management System XDM E2 Description*.
- If DCCM3 does not define the logical terminal name of the DCCM3 logical terminal that matches the terminal identification information specified on TP1/Client, the `dc_clt_connect_s` function returns a `DCCLTER_NET_DOWN` error.

2.2.4 Notes on using permanent connection

- You cannot establish permanent connection from within a transaction. After establishing permanent connection, generate a transaction. When you specify a DCCM3 logical terminal as the destination of a request to establish permanent connection, you cannot generate transactions from a CUP.
- When the facility for establishing permanent connection is used, if a communication error or timeout occurs on the client, the permanent connection will be released.
- If any timer used for monitoring on the server side expires during establishment of a permanent connection, the server may fail. If the server fails, a function issued by a CUP may be placed in a wait state until a timeout occurs (until the maximum time to wait for a response expires). This phenomenon occurs because the server cannot recognize the packets sent from the client. To prevent this phenomenon from occurring, set appropriate values in all the timers.

2.3 Remote procedure calls

This section describes the process of a remote procedure call (RPC) from a CUP to a SPP.

For details of RPCs, see the manual *OpenTP1 Programming Guide*.

2.3.1 RPC initiation

To request an SPP service, execute a service-requesting function from the CUP. Specifically execute the `dc_rpc_call_s` function whose arguments specify the SPP service group name and service name.

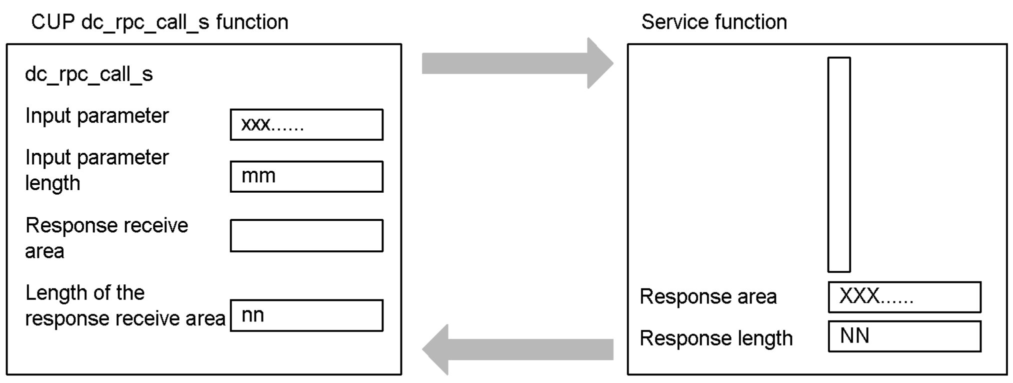
The SPP is activated by the TP1/Server user service configuration definition or the `dcsvstart` command.

2.3.2 RPC data transfer

To pass data in an RPC, specify the following arguments when executing the `dc_rpc_call_s` function: SPP service group name, service name, input parameter, input parameter length, service response storage area, and response length.

The following figure shows RPC data transfer.

Figure 2-2: RPC data transfer



Note: The response length set in the dc_rpc_call_s function (nn) should be the same length or longer than the response from the service function (NN).

2.3.3 RPC types

TP1/Client can execute synchronous response type RPCs or no-response type RPCs.

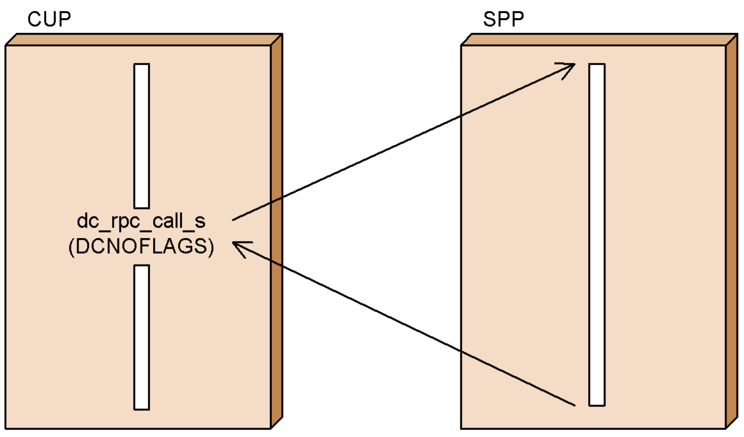
(1) Synchronous response type RPC

The CUP sends an inquiry message to an SPP and receives a response message.

Subsequent CUP processing waits until the SPP returns the processing result.

The following figure shows the process flow of a synchronous response type RPC.

Figure 2-3: Process flow of a synchronous response type RPC

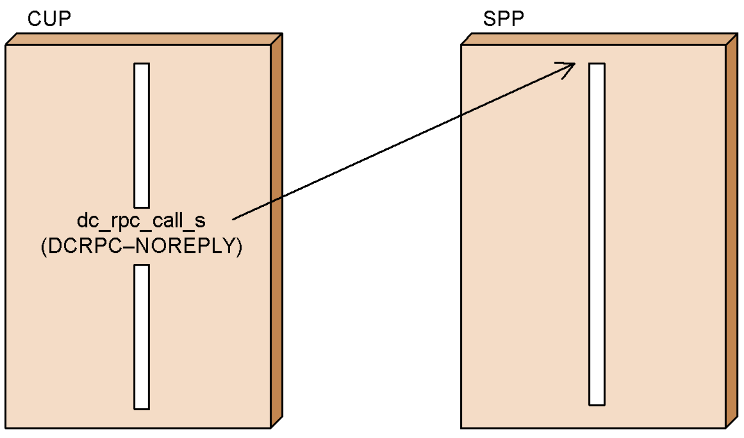


(2) No-response type RPC

The CUP sends an inquiry message to an SPP but does not receive a response message. Subsequent CUP processing is executed without receiving the processing result from the SPP.

The following figure shows the process flow of a no-response type RPC.

Figure 2-4: Process flow of a no-response type RPC



2.3.4 Chained RPC

A multi-server environment can concurrently activate multiple instances of the same

SPP for multiple processes. In this environment, an SPP execution process is started every time a service is requested. When a CUP calls the same service group more than once, the SPPs of the service group are not always executed with the same process. However, if you use synchronous-response RPCs to request more than one service of the same service group, the services can be executed with the same process. This execution method is called a *chained RPC*.

A chained RPC can be used only when a transaction is started from a CUP or a permanent connection has been established.

Using a chained RPC reduces the number of user processes required for processing one transaction, leading to a lower load on the transaction processing. If used as a transaction, the chained RPC works on one global transaction.

The chained RPC is assured on a CUP process basis. However, note that, if a different client UAP is used even within the same global transaction, it is not assured that the service called several times is started in the same process.

(1) Starting a chained RPC

To request a service that will work in chained-RPC mode, specify `'DCRPC_CHAINED'` in the flags of the `dc_rpc_call_s` function. Requesting the service with this value specified lets the SPP recognize the chained RPC and reserves a process. Specify `'DCRPC_CHAINED'` in the flags for the second or subsequent service requests.

(2) Terminating a chained RPC

A chained RPC can be terminated by either

- Issuing the `dc_rpc_call_s` function whose `flags` is `DCNOFLAGS` to the service group working in the chained-RPC mode,

  or

- Performing synchronous-point processing (commit or rollback) to complete the global transaction executing in chained-RPC mode.

(3) Watching chained RPC time

The UAP requested to provide a service in chained-RPC mode watches the time between returning a response to the CUP and requesting the next service or the synchronous-point process of a transaction. If the next service or synchronous-point processing request does not arrive within the watching time, the system assumes a CUP error and terminates the SPP abnormally. The watching time must be specified in `'watch_next_chain_time'` of the user service definition.

2.3.5 Scheduler

The TP1/Server Base scheduler is valid as well for a service request from the CUP to an SPP. TP1/Server Base creates a schedule queue for each SPP service group and schedules a service request. If the SPP is specified as a server that receives requests from socket (`'receive_from=socket'` specified by the TP1/Server Base user service definition), the SPP can directly receive a service request from the CUP, not via the schedule queue.

Multiple nodes can activate the server that processes the same service group. This server is called a *multi-node server*. It can distribute service requests from the client to each node, decreasing the processing load. This facility is called the *internode load-balancing*. However, distributing requests requires extra memory for storing server information. When you are sure the single server is operating, you can maintain improved performance by not distributing service requests.

`DCCLTLOADBALANCE` in the client environment definition specifies whether to use the internode load-balancing.

Specify the memory size for storing server information using `DCCACHE` in the client definition. When you specify a larger memory size, you can store more server information, distributing service requests to many servers. If you do not use the load distributing function, allocating more memory speeds up the service information retrieval, thus improving the performance.

When you use the load distributing function, you can specify an effective period for retaining the server information in memory. The shorter the effective period is, the more often the server load information is retrieved, distributing the load based on the most recent state. Because this frequently causes an overhead for load information retrieval, however, the performance may degrade. To specify the effective period, use `DCCLTCACHETIM` in the client environment definition.

2.3.6 Inter-node load-balancing facility

OpenTP1 has the *inter-node load-balancing facility* that distributes the load across nodes so that the RPC-based requests do not concentrate on a specific node.

To use the inter-node load-balancing facility, a load-balancing environment must satisfy the following conditions:

- A user server for providing the same services to more than one node is running.
- For the OpenTP1 nodes to exchange their user server information (name information) with each other, in each node, the other nodes are specified in the `all_node` clause in the system common definition.

This subsection explains the related definitions, processing, and RPC handling of the client side and the server side when using the inter-node load-balancing facility.

(1) Server-centric load balancing

The schedule service of TP1/Server distributes the load to the nodes that can execute more efficiently according to the schedule status of the nodes.

(a) Definition on the client side

Specify `dcscddirect=Y` in the client environment definition. By this specification, the clients request the load-balancing from the schedule service of TP1/Server. Therefore, the OpenTP1 node containing the governing schedule service must be defined on the client side.

The OpenTP1 nodes request scheduling in the order in which they are specified in the `dchost` operand. To make the OpenTP1 nodes request scheduling at random, you must also add `dchostselect=Y` in the definition.

(b) Definition on the server side

On the server side, configure either of the following settings:

- In the schedule service definition, set the following operands:

  `scd_this_node_first=N` (default)

  `scd_announce_server_status=Y` (default)
- Omit the schedule service definition.

(2) Client-centric load balancing by using the load information from the server

To use this feature, define the following:

- Definition on the client side

  Specify `DCCLTLOADBALANCE=Y` in the client environment definition.
- Definition on the server side

  Specify `scd_announce_server_status=Y` (default value) in the schedule service definition.

With this specification, the client acquires the server's load level from the server to determine the OpenTP1 node from which the client requests a service. The client then executes an RPC.

The client asks the name service of TP1/Server, which functions as a gateway for clients, for the information about the service. The client then temporarily stores the service information, including the server load level, in a cache area whose size is specified in the `DCCACHE` operand in the client environment definition.

If the applicable service information exists in the cache area when the client executes the RPC, the client does not ask the name service of TP1/Server that is used as a gateway for service information.

Since clients manage the cache using the Least Recently Used (LRU) method, service information is deleted starting from the least referenced information when the cache areas run short. Also, service information that has exceeded its effective period specified in the `DCCLTCACHETIM` operand in the client environment definition is deleted from the cache areas when RPCs are executed. When this happens, the clients ask the name service for service information.

When you increase the value specified for the `DCCACHE` operand in the client environment definition, a large amount of service information can be stored. This reduces the number of communications with the TP1/Server name service that is used as a gateway. However, since service information is retrieved from many cache areas, more overhead is required.

When you decrease the value specified for the `DCCACHE` operand in the client environment definition, the information about the services provided by applicable SPP on multiple nodes may exceed the cache areas. In such a case, even if you re-execute an RPC from the client, the RPC-based request is not sent to the SPP on the node whose service information is not stored in a cache area.

When you decrease the value specified for the `DCCLTCACHETIM` operand in the client environment definition, old service information is immediately deleted and the client asks the TP1/Server name service that is used as a gateway for new service information. Since new service information is always stored in the cache areas, RPC-based requests can be distributed in accordance with the load of the servers. However, the number of communications with the name service increases and the overhead for rewriting the cache areas also increases.

When you increase the value specified for the `DCCLTCACHETIM` operand in the client environment definition, the number of communications with the TP1/Server name service that is used as a gateway can be reduced. However, the reaction to status changes of SPPs is delayed, and RPC-based requests may be sent to an inactive SPP. In this case, TP1/Client deletes the applicable service information from the cache areas before you send an RPC-based request to another SPP. Then, TP1/Client sends a request to the TP1/Server name service that is used as a gateway to delete the applicable service information.

In a multi-node server configuration with 129 or more servers, when `nam_service_extend=1` is specified in the name service definition for the TP1/Server that is used as a gateway, specify `DCCLTNAMEXTEND=1` in the client environment definition. By doing so, you can increase the maximum number of service information items that a client can acquire at a time from the name service, from 128 to 512.

(3) Operations when the load-balancing facility is used with other facilities

Table 2-1 shows the operations when the load-balancing facility is used with other facilities.

Table 2-1: Operations when the load-balancing facility is used with other facilities

Condition	Operation
When a permanent connection is used in the client	The CUP executing process on the server side issues RPCs on the node to which a permanent connection is established. In this case, the load-balancing facility operates in the same way as in (2)(b) above.
When the transaction control API is used in the client	The alternate transaction-executing process on the server side issues RPCs. In this case, the load-balancing facility operates in the same way as in (2)(b) above.
When the remote API facility is used	The RAP-processing server on the server side issues RPCs. In this case, the load-balancing facility operates in the same way as in (2)(b) above.

2.3.7 RPC time monitoring

When you use a synchronous-response type RPC, you can monitor the time until a response message is received. Specify the monitoring time using `DCWATCHTIM` in the client environment definition.

You can also set the monitoring time by letting the CUP execute the `dc_rpc_set_watch_time_s` function.

To change the monitoring time depending on services you request, set the monitoring time before executing the RPC. You can reference the specified monitoring time by executing the `dc_rpc_get_watch_time_s` function.

When no response message returns after the specified monitoring time, the RPC returns control with an error.

2.3.8 Authentication RPC

You can execute an RPC for the server that is protected by the OpenTP1 security service. When executing the RPC, check to see if the user authenticated by the user authentication function has the right to request the service.

Using the authentication RPC function requires more memory to execute the CUP. When the memory usage is limited for CUP execution, or the RPC is executed for a server not protected by the security service, disabling the authentication RPC increases the free memory size. Specify `DCCLTSECURITY` in the client environment definition not to use the authentication RPC.

The authentication RPC function is unavailable if no security function is installed on TP1/Server that authenticated the user. No extra area is allocated in memory. You need not explicitly disable the use of the authentication RPC.

2.3.9 RPC to servers other than OpenTP1

TP1/Client can issue an RPC to non-OpenTP1 servers such as DCCM3. To implement this feature, the server needs a function that interprets OpenTP1 RPC requests.

(1) Specifying the remote server

For issuing an RPC, specify the remote server using the service group name and the service name in the same manner as RPCs for the OpenTP1 server. Since the client calls a server that is not controlled by the OpenTP1 name service, the client needs a feature that is equivalent to the name service for interpreting addresses.

(2) Address definition for the remote server

At the client side, create a text file for a list of RPC entry points (host computer names and port numbers) corresponding to the service group name. Declare this text file in `DCCLTSERVICEGROUPLIST` of the client environment definition.

At RPC execution, TP1/Client checks this list for the specified service group name. If a match is found, TP1/Client issues the RPC to the corresponding RPC entry point.

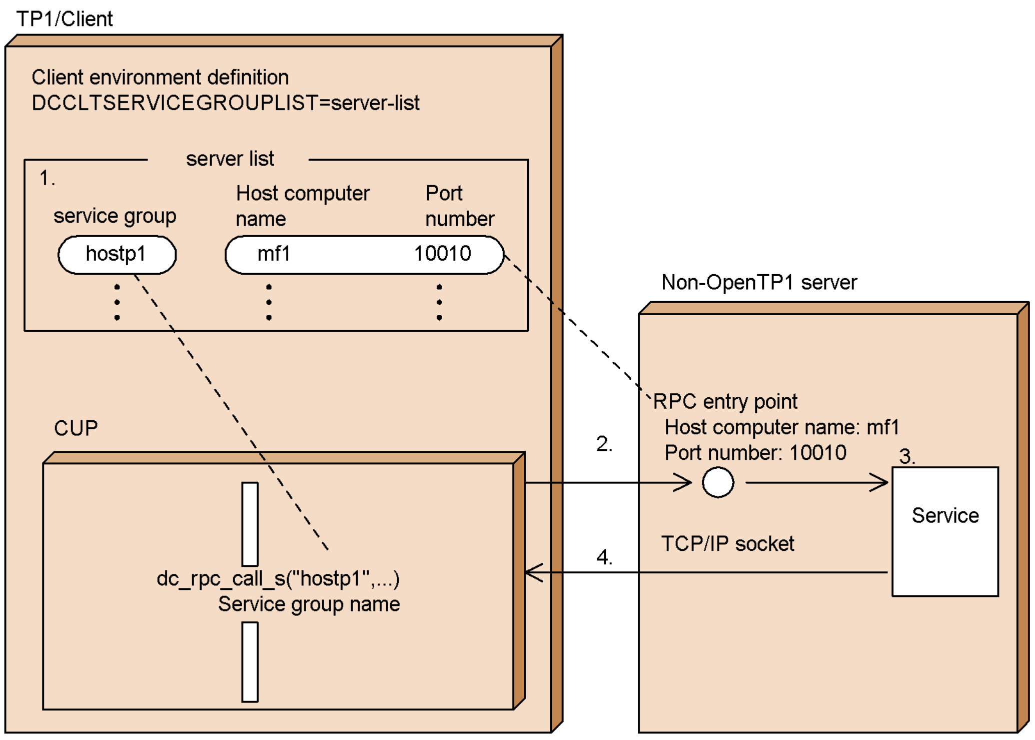
(3) RPC function overview

Available RPCs are of synchronous-response and non-response types. If an RPC is executed under transaction control, an OpenTP1 transaction does not process it. Because the service group name and the server makes a pair, the load distributing function is also unavailable. However, if you use a permanent connection to execute an RPC to a DCCM3 logical terminal, the load distributing function is available. For details on the load distribution when executing an RPC to a DCCM3 logical terminal, see (4).

When you issue an RPC to DCCM3, the service name is assumed to be a transaction name.

The following figure illustrates RPC processing for non-OpenTP1 server.

Figure 2-5: RPC processing for a non-OpenTP1 server



1. The client environment definition specifies correspondence between the service group name and the RPC entry point.
2. At RPC execution, the CUP finds the RPC entry point address from the service group name and sends an RPC message.
3. The server interprets the RPC message and executes the requested service.
4. A synchronous-reponse type RPC receives a response message from the server.

(4) Load distribution when executing an RPC for a DCCM3 logical terminal

When TP1/Client and the DCCM3 logical terminal use a permanent connection to execute an RPC, the load can be distributed to multiple DCCM3s. The system selects the connection destinations at random from the host names and port numbers of multiple DCCM3 logical terminals specified in the client environment definition, and the system attempts to connect the selected destinations. If an attempt to connect the selected DCCM3 logical terminals fails, those DCCM3 logical terminals are eliminated from the options. Then, the system re-selects some of the remaining DCCM3 logical terminals specified in the client environment definition at random, and the system attempts to connect the selected destinations. This step will be repeated. If all the attempts to connect the DCCM3 logical terminals specified in the definition fail, the system assumes an error.

The methods of communication possible between TP1/Client and the DCCM3 logical terminal are as follows. However, only methods 1 and 2 can perform load distribution since they use a permanent connection.

1. Specify the host name of the DCCM3 logical terminal in the `DCCLTDCCMHOST` client environment definition and the port number of the DCCM3 logical terminal in `DCCLTDCCMPORT`. Specify `DCCLT_DCCM3` in the flags argument of the `dc_clt_connect_s` function. A permanent connection is used in this case.
2. Specify the host name and the port number of the DCCM3 logical terminal in the `DCCLTRAPHOST` client environment definition and `DCNOFLAGS` in the flags argument of the `dc_clt_connect_s` function. A permanent connection is used in this case.
3. Specify the file that specifies the host name and the port number of the DCCM3 logical terminal for each service group in the `DCCLTSERVICEGROUPLIST` client environment definition. A permanent connection is not used in this case.

2.3.10 RPC using the name service

TP1/Client performs RPCs using the name service function of TP1/Server. TP1/Client manages OpenTP1 system service information using the name service function.

Since TP1/Client uses the cache for service information, it minimizes inquiries into the TP1/Server name service. If the cache contains no service information or the effective cache period expires, TP1/Client communicates with the TP1/Server name service.

To minimize communications between TP1/Client and the server, you can disable the TP1/Server name service function. In this case, an RPC generates no name service inquiry, decreasing communication load between TP1/Client and the server. This is useful when TP1/Client is connected to the server via WAN.

Specify whether to perform the RPC without the name service using `DCSCDDIRECT` of the client environment definition.

The name service function directly allows a remote service request such as the schedule service for `DCHOST` in the client environment definition or the host computer specified when the user authentication is executed. When this function is used, RPCs are unavailable for the socket-receiving type server SPP.

2.3.11 RPC using the multi-scheduler facility

When a CUP requests, a service of the SPP that uses the schedule queue (queue-receiving server), the scheduler daemon of the node that has the requested SPP receives the service request message and stores the message into the schedule queue of the requested SPP. The scheduler daemon is a system daemon that provides schedule services.

When a long service request is sent to the scheduler daemon, the request is divided into parts of a fixed length. The scheduler daemon then assembles the parts into the original request, and stores the assembled message in the schedule queue of the queue-receiving server. In an OpenTP1 system, multiple scheduler daemon processes cannot be executed concurrently. Therefore, while the scheduler daemon is receiving a service request message, the scheduler daemon cannot receive any other service request messages. If a long service request is sent over a slow line, scheduling of other service request messages might be delayed, hindering the streamlining of the scheduling in a large-scale system made up of high-performance machines and networks. To prevent scheduling delays, in addition to the scheduler daemon, you can start multiple daemon processes dedicated to receiving service requests to receive service request messages in parallel. This functionality is called the *multi-scheduler facility*. Hereafter, the ordinary scheduler daemon is called the *master scheduler daemon*, and the daemons dedicated to receiving service requests are called *multi-scheduler daemons*.

For details about system configurations that might require the multi-scheduler facility, see the manual *OpenTP1 Version 7 Programming Guide*.

(1) Selecting a multi-scheduler daemon at random

The multi-scheduler facility randomly selects one of the active multi-scheduler daemons to send service requests.

A multi-scheduler daemon can be selected at random to execute the following RPCs:

(a) RPCs that do not use the name service

When you specify `Y` in the `DCSCDDIRECT` client environment definition to execute an RPC that does not use the name service, the daemon selection method differs depending on whether `DCSCDPORT` is specified in the client environment definition.

- `DCSCDPORT` is specified

  When either of the following port numbers is specified in `DCSCDPORT`, the system can randomly select a multi-scheduler daemon without inquiring the service information from the name service of the TP1/Server that is assigned as a gateway. This eliminates the communication with the name server, thus reducing the load on the name service.

  - Port number of the schedule service specified in the `scd_port` operand in the schedule service definition
  - Port number specified in the `-p` option of the `scdmulti` schedule service definition

  Beforehand, you must specify the number of multi-scheduler daemon processes started on TP1/Server in `DCSCDMULTICOUNT` of the client environment definition. The port number of a multi-scheduler daemon used to send service requests is selected at random from the following range of port numbers:

  - Lower limit: Port number value specified in `DCSCDPORT` of the client environment definition

- Upper limit: Lower limit value + the number of processes specified in `DCSCDMULTICOUNT` of the client environment definition - 1

Note that the value specified in the `scdmulti` schedule service definition must be consistent with TP1/Server gateways that are specified in the `DCHOST` client environment definition.

■ `DCSCDPORT` is not specified

The system requests the authentication of the client user. After the client user is authenticated, the system obtains the service information from the name service of TP1/Server which is assigned as a gateway at the first service request. The system uses this information to select multi-scheduler daemons at random and sends the service requests. The service information is valid until either of the following conditions applies:

- The authentication of the client user is disabled (by executing the `dc_clt_cltout_s` function).
- The gateway TP1/Server is switched.

The range of port numbers of multi-scheduler daemons for sending service requests is determined based on the port number specified in the `-p` option which is specified first in the `scdmulti` schedule service definition. Therefore, consider the sequence of specifying the schedule service definition.

(b) Regular RPCs

This subsection applies when the multi-scheduler facility is used to execute regular RPCs (RPCs that use the name service). In this case, if the temporary storage area for service information does not contain the desired service information, the system asks the name service for the service information. The system uses the service information to select multi-scheduler daemons at random and sends service requests.

(2) Client environment definition and scheduler daemon for sending service requests

When you use the multi-scheduler facility, the scheduler daemon for sending service requests differs depending on the specification in the client environment definition.

Table 2-2 shows the relationship between the client environment definition and the scheduler daemon.

Table 2-2: Client environment definition and scheduler daemon

Operand values in the client environment definition				Scheduler daemon for sending service requests
DCSCDMULTI	**DCSCDDIRECT**	**DCSCDPORT**	**DCSCDMULTI-COUNT**	
Y	Y	V	V	Multi-scheduler daemon selected at random[#1]
			--	Scheduler daemon started with the port number specified in `DCSCDPORT`
		--	V	Multi-scheduler daemons randomly selected from the multi-scheduler daemons that are determined based on the port number specified in the `-p` option defined first in the `scdmulti` schedule service definition[#2]
			--	
	N	V	V	Multi-scheduler daemon randomly selected based on the service information[#2]
			--	
		--	V	
			--	
N	Y	V	V	Scheduler daemon started with the port number specified in `DCSCDPORT`
			--	
		--	V	Master scheduler daemon[#2]
			--	
	N	V	V	Server that receives requests from a socket or master scheduler daemon[#2]
			--	
		--	V	
			--	

Legend:

Y: `Y` is specified for the operand.

N: `N` is specified for the operand.

V: A value is specified for the operand.

--: No value is specified for the operand.

#1:

The port number of a multi-scheduler daemon is selected from the following range:

Lower limit: Schedule service port number specified in `DCSCDPORT` of the client environment definition

Upper limit: Lower limit value + the `DCSCDMULTICOUNT` value in the client environment definition - 1

#2:

Inquiries to the name service occur.

2.3.12 Switching facility of TP1/Server as a gateway

If an error occurs in TP1/Server as a gateway, after the error is returned, the definition of TP1/Server as the gateway needs to be changed. However, if multiple TP1/Servers are specified as gateways, TP1/Client switches to the next specified TP1/Server if an error occurs.

When CUPs request service at the same time, inquires will become concentrated on the name service of a single gateway TP1/Server, thus increasing the load. The random selection of multiple TP1/Servers distributes the load on the gateway TP1/Server. For details on the load distribution by random selection of multiple TP1/Servers, see Subsection *2.3.13 Load distribution for TP1/Server as a gateway*.

(1) TP1/Server switches because of the following conditions.

- If an error is detected in TP1/Server as a gateway when performing user authentication.
- If the communication fails with the name service of TP1/Server, which is used as a gateway when issuing RPCs.
- If the communication fails with the schedule service of TP1/Server, which is used as a gateway when issuing the RPC without the name service.
- If the communication fails between the transaction accept service of TP1/Server when issuing the RPC that allocates the transaction management process.
- If the client receives an error response from TP1/Server used as a gateway while a service is being requested, a schedule service is being started, or a schedule service is being terminated.

(2) How to specify the switch destination OpenTP1

- Specify multiple TP1/Servers, which are used as gateways, in the `target_host` argument of the user authentication function.

- Specify multiple TP1/Servers, which are used as gateways, in `DCHOST` of the client environment definition.
- In addition to the above specifications, specify `DCHOSTCHANGE=Y` in the client environment definition to switch TP/Server when:

  The client receives an error response from TP1/Server used as a gateway while a service is being requested, a schedule service is being started, or a schedule service is being terminated.

(3) Sequence of switching

- When `target_host` is specified as an argument of the user authentication function, TP1/Servers switch in the sequence they are specified in `target_host`.
- If `target_host` is not specified as an argument of the user authentication function, TP1/Servers switch in the sequence specified in `DCHOST` in the client environment definition.

2.3.13 Load distribution for TP1/Server as a gateway

When CUPs request service at the same time, inquires will become concentrated on the name service of a single gateway TP1/Server, thus increasing the load. If TP1/Client can select multiple TP1/Servers at random as gateways, the load on the gateway TP1/Server can be distributed.

(1) Selecting gateway TP1/Servers at random

To select gateway TP1/Servers at random, specify `Y` in the `DCHOSTSELECT` client environment definition.

When a CUP requests service, the CUP inquires the node that has the desired service from the name service of the gateway TP1/Server which is specified in the `DCHOST` client environment definition.

When multiple gateway TP1/Servers are specified, TP1/Client first inquires of the name service of the gateway TP1/Server that is specified at the beginning. If the service request is not accepted since TP1/Server specified at the beginning is not started or for other reasons, an attempt is made to switch to the next TP1/Server which is specified as the next gateway. This is the only case where a TP1/Server switch is attempted.

(2) Distributing the load on the gateway TP1/Server with priority

TP1/Server distributes the service requests to multiple nodes to distribute the load on the nodes. The schedule service of a node that receives service requests may sometimes transfer the service requests to TP1/Server on a different node to distribute the load. However, if TP1/Client has selected gateway TP1/Servers at random, the service requests are distributed from the already-selected gateway TP1/Server to a different TP1/Server in a different node. This may cause some overhead.

To prevent the above situation, you can distribute the load on the gateway TP1/Server selected by TP1/Client with priority. To distribute the load on the gateway TP1/Server with priority, specify `Y` in the `DCSCDLOADPRIORITY` client environment definition.

This definition is valid only when you execute RPCs without using the name service (specify `Y` in the `DCSCDDIRECT` client environment definition).

Note

When you distribute the load on the gateway TP1/Server of a node with priority and the gateway TP1/Server stops due to an error, TP1/Client switches the failed gateway TP1/Server to a different gateway TP1/Server. Even if the previous TP1/Server is restarted, TP1/Client uses the current gateway TP1/Server with priority. Therefore, the restarted TP1/Server may not receive as many service requests as before from TP1/Client even though its load is low.

To change the gateway TP1/Server, reexecute the canceling of the authentication of the client user (`dc_clt_cltout` function or `dc_clt_cltout_s` function) and the requesting of the authentication of the client user (`dc_clt_cltin_s` function). By doing so, the restarted TP1/Server may be assigned as the gateway TP1/Server .

2.3.14 Data compression

The data compression compresses the user data sent to the network by RPCs. Compression reduces the number of packets sent to the network, easing congestion in the network.

(1) Specification method

Specify data compression using `DCCLTDATACOMP` in the client environment definition.

(2) Scope

The data compression can be used if the service-requested TP1/Server Base is version 03-03 or later.

(3) Overview of the facility

Using this facility, TP1/Client compresses the value of the input parameter (`in`) set by the `dc_rpc_call_s` function which is issued from the CUP and sends it to the network. For this inquiry, the value of the response (`out`) returned from the SPP is also compressed by TP1/Server and sent to the network. TP1/Client that receives the response decompresses the compressed data and passes it to the CUP.

The compression depends on the version of the service-requested TP1/Server Base as follows.

(a) Version 03-05 or earlier

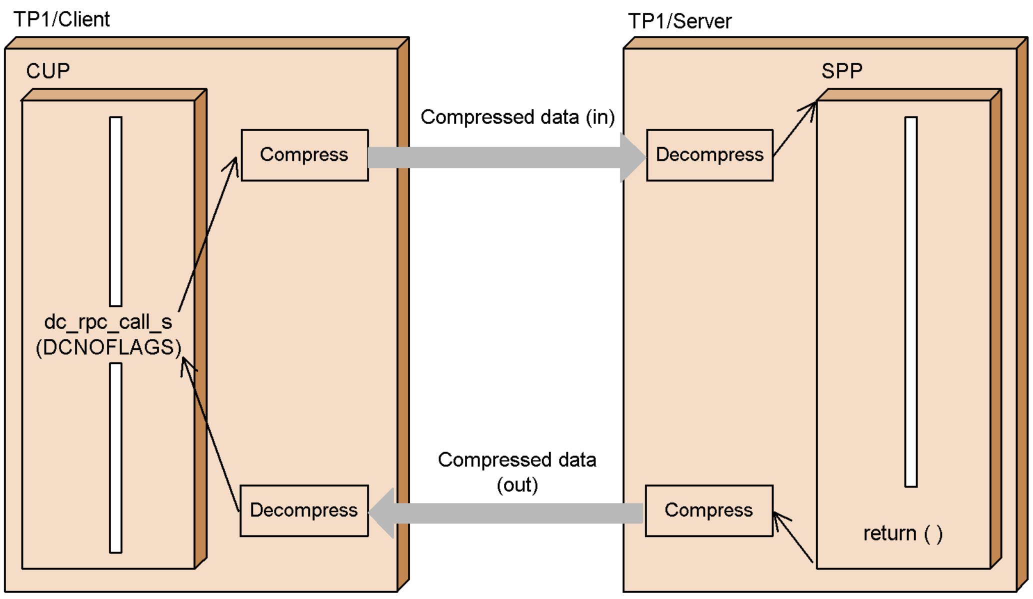
When the input parameter value is not compressed, TP1/Server Base does not compress the response value even if compressing it will be effective.

(b) Version 03-06 or later

Even if the input parameter value is not compressed, TP1/Server Base compresses the response value if compressing it will be effective.

The following figure gives an overview of the data compression.

Figure 2-6: Overview of data compression



Note: Even if the input (in) is not compressed, TP1/Server Base with version 03-06 or later compresses the output (out) if compressing it will be effective.

(4) Effect of the data compression

The effect of the data compression depends on the contents of the user data. The data compression is effective when the user data contains many consecutive identical characters. For some types of user data, it is hardly effective at all. When executing the `dc_rpc_call_s` function more than once from the same CUP, consider the effect of the data compression for each CUP.

Because compressing and decompressing data requires much overhead, consider the effect of the data compression and evaluate its performance before using it.

2.3.15 Remote API facility

TP1/Client provides permanent connection between a CUP and the server. Using this connection, the CUP can transfer an API to the server, allowing a server process to execute that API. This capability is called the remote API facility. You can also use the remote API facility to issue a service request to a UAP inside a firewall. A firewall, which is located between a shared network and a restricted network, is hardware and software for preventing a third party from illegally entering the network.

When passing through the firewall using the remote API facility, the CUP issues the `dc_clt_cltin_s` (`DCCLT_NO_AUTHENT` specified in flags) function to send a request for establishing a permanent connection to the RAP-processing listener in TP1/Server.

Upon the reception of a response for establishing the permanent connection, the permanent connection with the RAP-processing server is established. After the establishment of a permanent connection, any request to the RAP-processing server is sent using the permanent connection. After disconnection, however, the system sends a request to the RAP-processing listener.

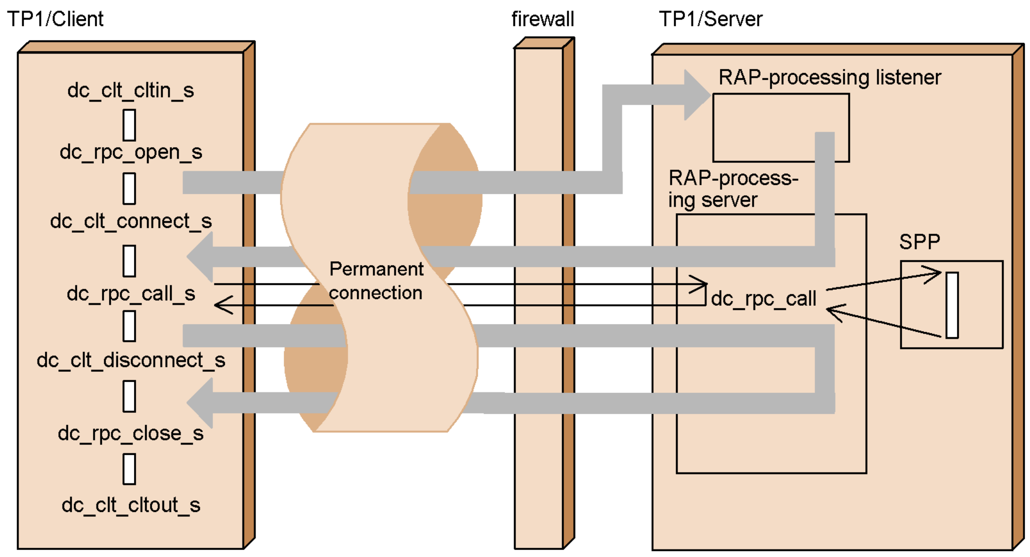
When using this function, specify `DCCLTRAPHOST` for the client environment definition.

The remote API facility is also available for requesting services to DCCM3 logical terminal.

When `Y` is specified in `DCCLTRAPAUTOCONNECT` of the client environment definition, a permanent connection is automatically established between the CUP and the RAP-processing server or between the CUP and the DCCM3 logical terminal. In this case, the `dc_clt_connect_s` function and the `dc_clt_disconnect_s` function do not need to be executed.

The following figure illustrates the remote API facility.

Figure 2-7: Remote API facility



(1) Prerequisites

The remote API facility is available for service-requested TP1/Server Base with version 03-05 or later.

To use the remote API facility to control transactions, service-requested TP1/Server Base must be version 03-06 or later.

You can request services to DCCM3 logical terminals with version 09-02 or later as well as:

- TP1/Client/W 03-04 or later
- TP1/Client/P 03-04 or later

(2) Scope

When using this facility to execute an RPC for DCCM3, you cannot generate a transaction for which the CUP is the root transaction. Also, you cannot concurrently establish a permanent connection with two or more remote systems.

This facility cannot access the XATMI interface.

(3) Selecting a permanent connection destination

The destination of a permanent connection varies with the flags set by the `dc_clt_connect_s` function, and with specified `DCCLTDCCMHOST` and `DCCLTRAPHOST` in the client environment definitions. When passing through the firewall using the remote API facility, specify `DCCLTRAPHOST` in the client environment definition.

Table 2-3 lists relationships between the function settings, definition specification and permanent connection destinations.

Table 2-3: Function settings, definition specification and permanent connection destinations

Argument flags	**Client environment definition**		**Establishing permanent connection for:**
	DCCLTDCCMHOST	**DCCLTRAPHOST**	
`DCNOFLAGS`	Y	Y	RAP-processing server or DCCM3 logical terminal[1]
		-	CUP execution process
	-	Y	RAP-processing server or DCCM3 logical terminal[1]
		-	CUP execution process
`DCCLT_DCCM3`	Y	Y	DCCM3 logical terminal[2]
		-	DCCM3 logical terminal[2]
	-	Y	Error return
		-	Error return

Legend:

Y: Specified

-: Not specified

1

Establish permanent connection for a DCCM3 logical terminal specified by `DCCLTRAPHOST`.

2

Establish permanent connection for a DCCM3 logical terminal specified by `DCCLTDCCMHOST`.

(4) Notes on controlling transactions by using the remote API facility

- The permanent connection destination you specify in the `DCCLTRAPHOST` client environment definition must be TP1/Server Base 03-06 or later. If you use a version earlier than this, the `dc_trn_begin_s` function returns a `DCCLTER_PROTO` error. This is also the same when the DCCM3 logical terminal is specified as the permanent connection destination. In this case, or the `dc_trn_begin_s` function returns a `DCCLTER_PROTO` error.

- The online tester facility of TP1/Server is unavailable. If you specify a test user ID in the `DCUTOKEY` client environment definition, the `dc_trn_begin_s` function returns an error with `DCCLTER_PROTO`.

2.3.16 Reducing server loads during timeout at synchronous response type PRC

When you execute the `dc_rpc_call_s` function from the TP1/Client CUP, TP1/Server accepts a service request. Since execution wait time, execution time, communication error or other causes may delay this request. TP1/Client sets the limit of response wait time to monitor errors.

TP1/Server does not have a way to handle the maximum response wait time for TP1/Client. TP1/Server may, therefore, continue processing even if TP1/Client has detected a timeout.

A function for reducing server loads during the timeout of a synchronous response type RPC allows TP1/Server to remove the above unnecessary processing.

(1) Specification

Use `DCWATCHTIMERPCINHERIT` in the client environment definition to specify whether to use the function for reducing server loads during the timeout of a synchronous response type RPC.

(2) Scope

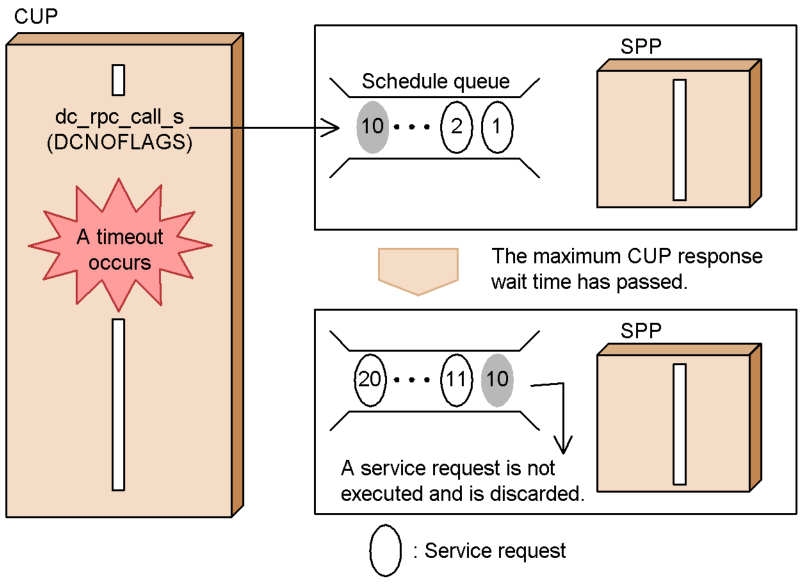
You can use this function when a 03-05 or later version of TP1/Server Base is the receiver of a service request.

(3) Function overview

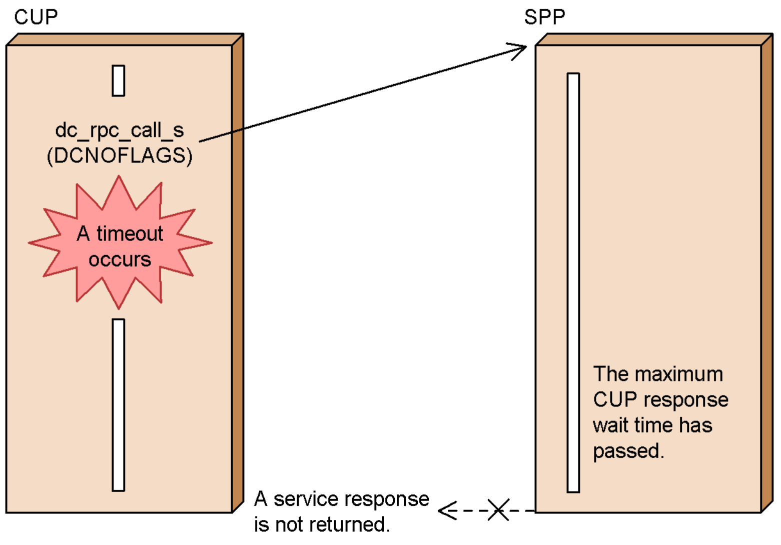
The following figure gives an overview of the processing by the function for reducing server loads during the timeout of a synchronous response type RPC.

Figure 2-8: Overview of processing by function for reducing server loads during timeout of synchronous response type RPC

(a) Example of discarding a service request for load reduction in the server


CUP
dc_rpc_call_s
(DCNOFLAGS)
A timeout occurs
SPP
Schedule queue
10
2
1
The maximum CUP response wait time has passed.
SPP
20
11
10
A service request is not executed and is discarded.
: Service request

(b) Example of not returning a service response for load reduction in the server


CUP
SPP
dc_rpc_call_s
(DCNOFLAGS)
A timeout occurs
The maximum CUP response wait time has passed.
A service response is not returned.

2.4 Transaction control

The CUP can issue the function that controls a transaction. The SPP to be executed as a transaction must have `"atomic_update=Y"` specified in the user service definition.

For details of transaction control, see the manual *OpenTP1 Programming Guide*.

This facility can be used when the version of TP1/Server Base is 03-00 or later.

2.4.1 Starting a transaction and acquiring a synchronous point

The CUP issues the `dc_trn_begin_s` function to start a transaction.

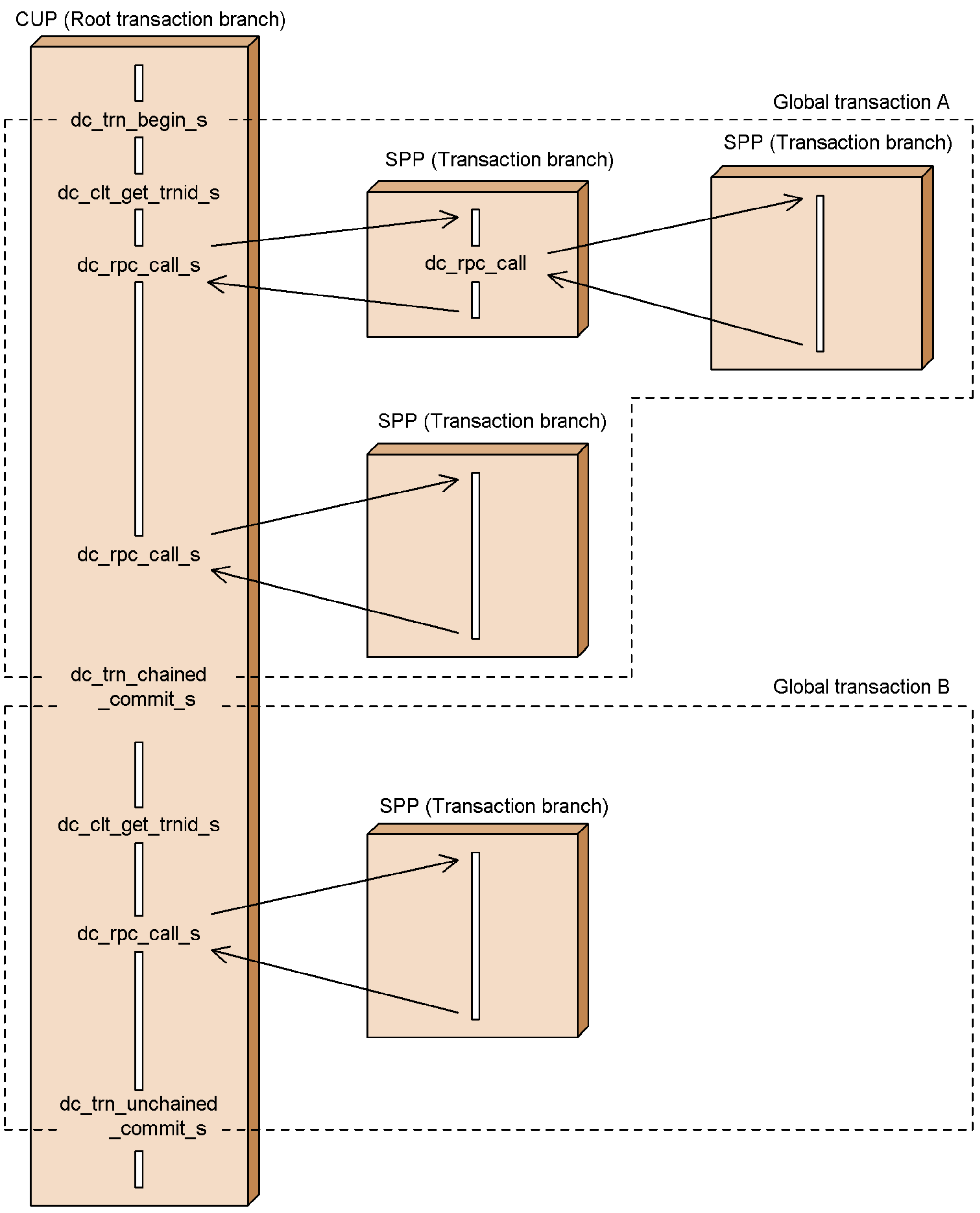
The range of a global transaction is from issuing the `dc_trn_begin_s` function to acquisition of a synchronous point (commit).

Once the `dc_trn_begin_s` function has been issued, another issuance of that function is unavailable within the global transaction.

If an RPC is executed from the CUP to an SPP, the CUP becomes a route transaction branch and the SPP a transaction branch.

The following figure shows the relation between transactions and RPCs.

Figure 2-9: Relationship between transactions and RPCs



2.4.2 Acquiring a synchronous point

(1) Commit

Synchronous point acquisition (commit) at the normal transaction termination is activated by the issuance of commit request functions. The global transaction is normally terminated when all transaction branches terminate normally.

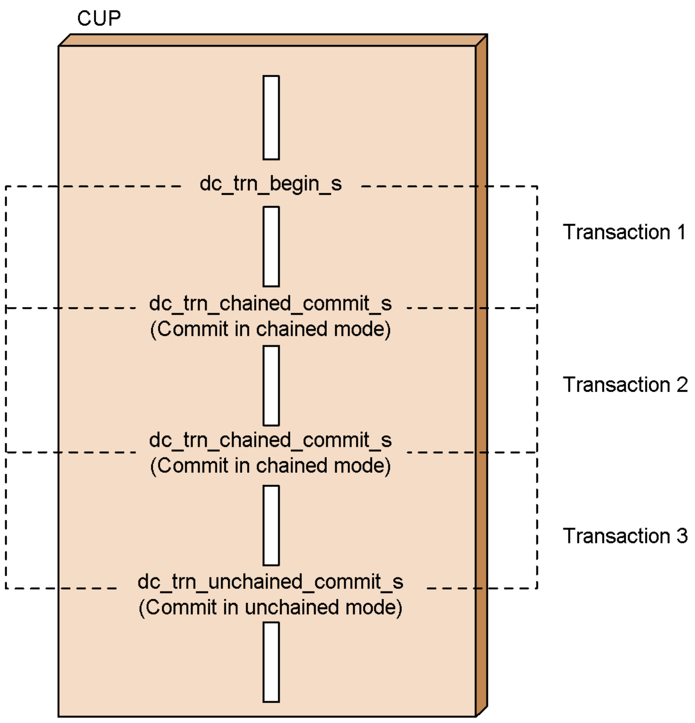
(a) Commit in chained/unchained mode

There are two modes where a synchronous point is acquired in transaction processing. In chained mode, one transaction terminates, followed by startup of another transaction at an acquired synchronous point. In unchained mode, one transaction terminates and a synchronous point is acquired but no new transaction takes place.

A commit in chained mode is called by the `dc_trn_chained_commit_s` function. A commit in unchained mode is called by the `dc_trn_unchained_commit_s` function.

The following figure shows transactions in chained and unchained modes.

Figure 2-10: Transactions in chained and unchained modes



(b) Processing where no commit request function is issued

If either of the following occurs, the transaction is rolled back:

- The program terminates without calling a commit request function and without issuing the `dc_rpc_close_s` function or `dc_clt_cltout_s` function.
- The CUP terminates abnormally before issuing a commit request function.

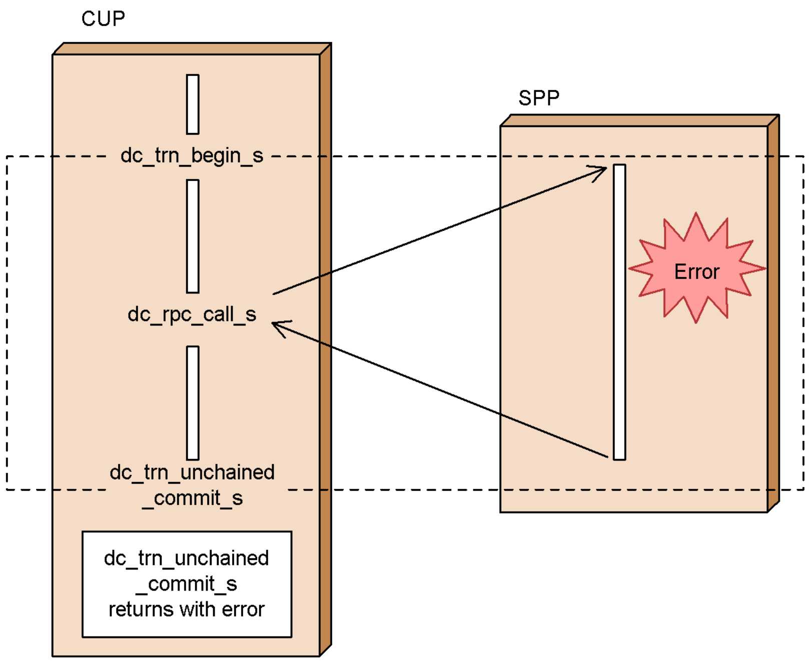
(2) Rollback

(a) Transaction errors in OpenTP1 processing

If an error occurs in a transaction, the commit request function is returned with an error and the transaction is rolled back for partial recovery. If an error occurs in any transaction branch of a global transaction, the entire global transaction is rolled back. Assuming that the transaction branch is to be rolled back, OpenTP1 executes partial recovery processing.

The following figure shows a transaction that is rolled back when an error occurs in OpenTP1 processing.

Figure 2-11: Rollback of transaction (when an error occurs in OpenTP1 processing)


CUP
SPP
dc_trn_begin_s
dc_rpc_call_s
Error
dc_trn_unchained _commit_s
dc_trn_unchained _commit_s returns with error

Note: The transaction processing enclosed by broken lines is rolled back.

(b) Issuing a rollback request function

To roll back a transaction at CUP's discretion, issue a rollback request function from the CUP.

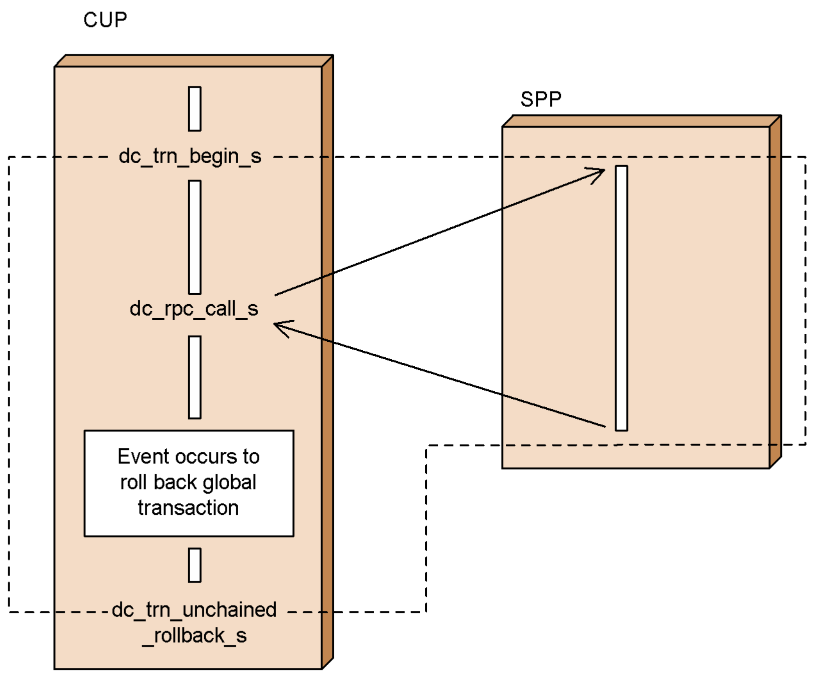
There are two modes of rollback: rollback in chained mode and rollback in unchained mode.

A rollback in chained mode is requested by issuing the `dc_trn_chained_rollback_s` function. In this rollback, the process of the CUP that issued the function remains in the global transaction after the rollback processing.

A rollback in unchained mode is requested by issuing the `dc_trn_unchained_rollback_s` function. In this rollback, the process that issued the function comes out of the global transaction after the rollback processing.

The following figure shows a transaction that is rolled back by a rollback request function.

Figure 2-12: Rollback of transaction (when a rollback request function is issued)


CUP
SPP
dc_trn_begin_s
dc_rpc_call_s
Event occurs to roll back global transaction
dc_trn_unchained _rollback_s

Note: The transaction processing enclosed by broken lines is rolled back.

(3) Disposal in case of heuristic situation

If a heuristic situation occurs in transaction processing, an error return takes place when the CUP acquires a synchronous point. The return values are:

- `DCTRNER_HEURISTIC (-3403)` if the result of deciding the heuristic situation does not match the result of the synchronous point of the global transaction
- `DCTRNER_HAZARD (-3404)` if a fault makes it impossible to identify the result of the synchronous point of the heuristically completed transaction branch

For the cause of sending back these return values and the result of the synchronous point of a global transaction, see the message log file.

For details of the disposal to be taken in the case of a heuristic situation, see the manual *OpenTP1 Programming Guide*.

(4) Transaction processing times

The following times regarding transactions can be specified by the client environment definition. For details, see Section *7.2 Definition details*.

- Expiry time in transaction branch
- Whether the following time is to be included in the monitoring time in transaction branch: the time required for the transaction branch being monitored to wait until the processing of another transaction branch called by the RPC function is completed
- Maximum time interval in transaction inquiry response
- CPU monitoring time in transaction branch

(5) Collection type for statistics of transaction branch

The client environment definition can specify the collection type for the statistics of a transaction branch. For details, see Section *7.2 Definition details*.

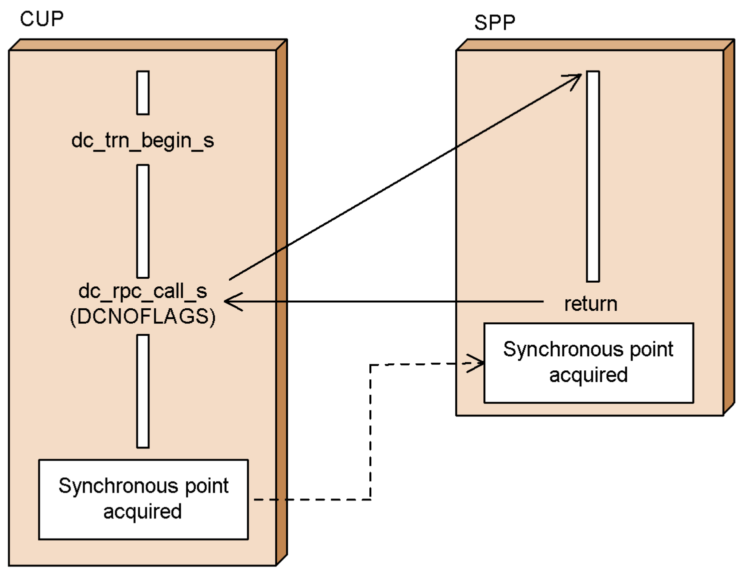
2.4.3 Relationship between remote procedure call modes and synchronous points

(1) Relationship between synchronous-response type RPCs and synchronous points

A synchronous-response type RPC transaction terminates when the result of its processing returns to the CUP and synchronous-point-acquisition processing ends.

The following figure shows the relationship between synchronous-response type RPCs and synchronous points.

Figure 2-13: Synchronous-response type RPCs and synchronous points

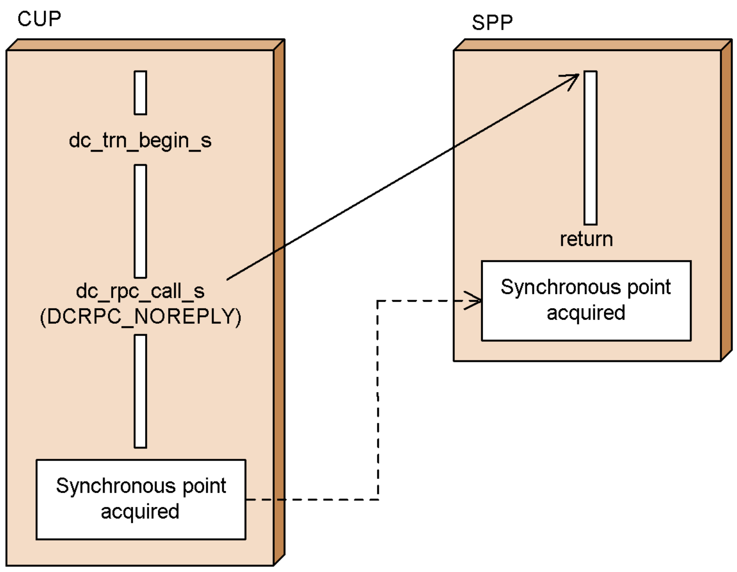


(2) Relationship between no-response type RPCs and synchronous points

In a transaction using a no-response type RPC, the CUP waits at a synchronous point until the SPP terminates, and then performs synchronous point processing.

The following figure shows the relationship between no-response type RPCs and synchronous points.

Figure 2-14: No-response type RPCs and synchronous points



(3) Relationship between chained RPCs and synchronous points

A chained RPC is executed with one SPP process. Therefore, the number of transaction branches is 1 regardless of how many times a chained RPC is used.

A transaction using a chained RPC terminates and the SPP process that executed the chained RPC is released when synchronous point processing finishes.

If a non-transactional chained RPC is used during a transaction, normally, the SPP process that executed the chained RPC is released when synchronous point processing finishes. If you want to release the SPP process with a synchronous-response type RPC without releasing it when synchronous point processing finishes, specify `00000002` in the `rpc_extend_function` operand of the user service definition.

The following figures show the relationship between chained RPCs and synchronous points.

Figure 2-15: Chained RPCs and synchronous points (transactional chained RPC)

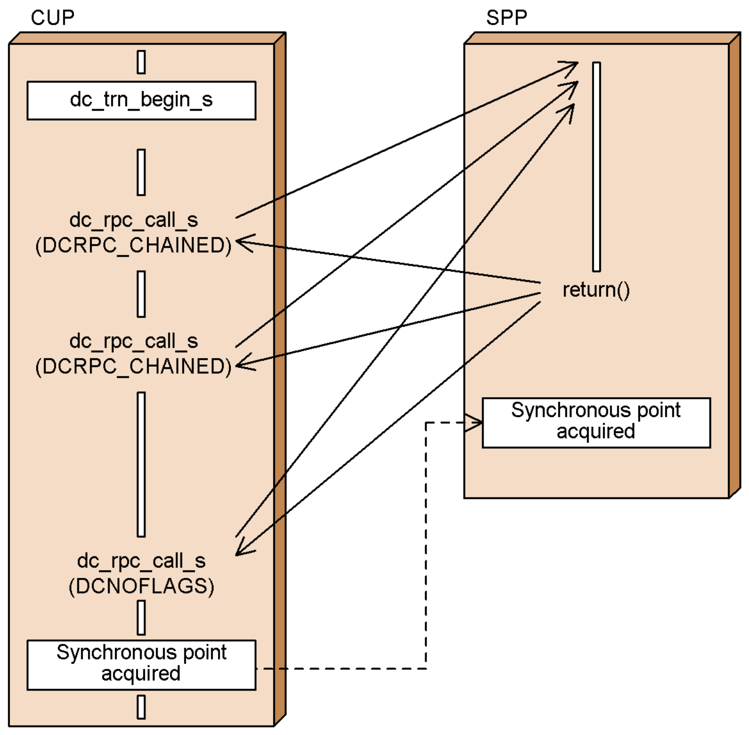
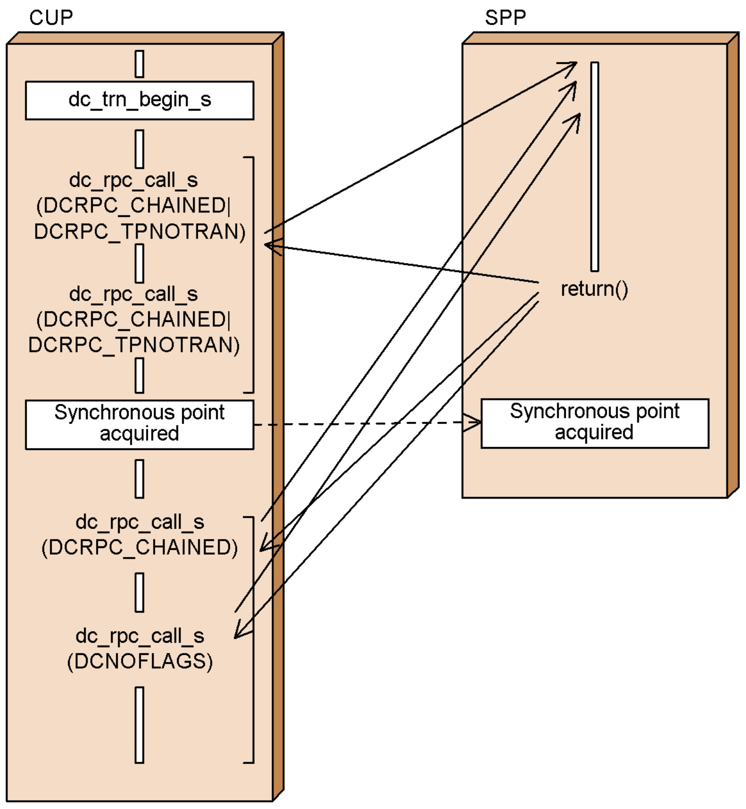

CUP
SPP
dc_trn_begin_s
dc_rpc_call_s (DCRPC_CHAINED)
dc_rpc_call_s (DCRPC_CHAINED)
return()
Synchronous point acquired
dc_rpc_call_s (DCNOFLAGS)
Synchronous point acquired

Figure 2-16: Chained RPCs and synchronous points (non-transactional chained RPC when no-release is specified)



2.4.4 Collecting identifiers for current transactions

By issuing the `dc_clt_get_trnid_s` function, the CUP can collect the current transaction global identifiers and transaction branch identifiers.

A transaction global identifier is needed to check whether the transaction started from the CUP has been committed when an error occurs. After any of the following functions is issued for possible errors, the `dc_clt_get_trnid_s` function must be issued:

- `dc_trn_begin_s` function
- `dc_trn_chained_commit_s` function
- `dc_trn_chained_rollback_s` function

2.4.5 Posting information for current transactions

The `dc_trn_info_s` function issued from the CUP makes it possible to use a return value for checking whether a transaction is active.

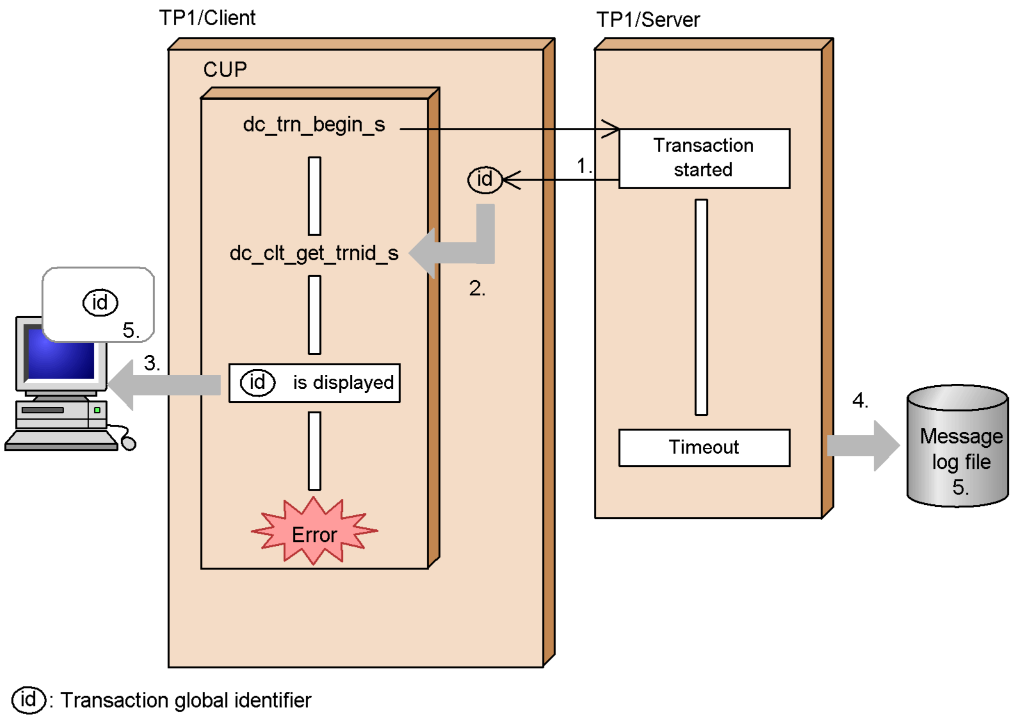
2.4.6 Detecting the synchronous point of a transaction when an error occurs

If an error occurs in a transaction started from the CUP, it can be detected whether its transaction branch has been committed. To detect whether the transaction branch has been committed, you must execute the `dc_clt_get_trnid_s` function to acquire the current transaction global identifier and transaction branch identifier after the transaction has started.

The transaction global identifier collected by the CUP is compared with the result of the transaction output to the server-side message log file to detect whether the transaction started from the CUP has been committed. The contents of the message log file can be displayed by the `'logcat'` command. For the `'logcat'` command, see the manual *OpenTP1 Operation*.

The following figure shows the detection method for the synchronous point of a transaction in the event of an error.

Figure 2-17: Detection method for synchronous point of transaction in the event of error



Explanation:

1. When the CUP issues the dc_trn_begin_s function to start a transaction, OpenTP1 reports its transaction global identifier to TP1/Client.
2. The CUP issues the dc_clt_get_trnid_s function to collect the transaction global identifier.
3. The collected transaction global identifier is displayed.
4. If an error occurs with the CUP and a timeout occurs on the OpenTP1 side, the result of processing the transaction is output to the message log file.
5. The transaction global identifier displayed in step 3 is compared with the contents of the message log file for matching.

2.4.7 Notes on transaction control

- If a timeout occurs within the scope of the transaction, the next function may return with a `DCCLTER_OLTF_NOT_UP` or `DCRPCER_OLTF_NOT_UP` error. This phenomenon occurs when a timeout occurs on the client side while the server is executing the function, and the client is disconnected from the server.

  This phenomenon does not occur when the value of `DCWATCHTIM` in the client environment definition is larger than the `watch_time` value of the transactional RPC executing process.

  The method of specifying the `watch_time` value differs depending on the version of TP1/Server Base as shown below:

- TP1/Server Base 03-02 or earlier:

  Use the `watch_time` operand in the system common definition or user service default definition.

- TP1/Server Base 03-03 or later:

  Use the `watch_time` operand in the client service definition.

You can also prevent this phenomenon from occurring by specifying `DCWATCHTIMINHERIT` and `DCCLTDELAY` in the client environment definition.

- If a transaction is started from a CUP, always commit the transaction in the unchained mode or issue a rollback request before terminating the CUP. If you terminate the CUP without terminating the transaction, you cannot check the result of the transaction at a synchronous point. In this case, the transactional RPC executing process is placed in a running state until a timeout for the transaction branch or a timeout for transaction inquiry occurs. When a timeout occurs, the transaction is rolled back.

  If you terminate the CUP or execute the `dc_clt_cltout_s` function without requesting a commit in the unchained mode or without requesting a rollback, a commit in the unchained mode is automatically performed. If this occurs, the CUP is not notified of the result at a synchronous point.

- After a timeout for real-time monitoring occurs, if you execute the `dc_rpc_call_s` function from a CUP, the function may return with a `DCRPCER_SYSERR` error.

- If any of the timers used for monitoring on the server side expires, the server may fail. If the server fails, a function issued by a CUP may be placed in a wait state until a timeout occurs (until the maximum time to wait for a response expires). This phenomenon occurs because the server cannot recognize the packets sent from the client. To prevent this phenomenon from occurring, set appropriate values in all the timers.

2.5 TCP/IP communication function

This section explains the sending and receiving of messages through the use of the TCP/IP communication function.

Note:

By using the TCP/IP communication function, you can communicate with MHPs. In this section, the remote system you communicate with is referred to as *MHP*. However, you can also freely select remote systems other than MHPs.

There are three types of message transmission by the TCP/IP communication function:

- Send-only messages from the CUP to the MHP
- Receive-only messages from the MHP to the CUP
- Send and receive messages between the MHP and the CUP

When a message is sent or received, automatic addition of the message length or confirmation of message delivery can be specified.

2.5.1 Send-only messages

The CUP can send a message to the MHP in send-only mode. This is called a *send-only message*.

To send a message to the MHP, the CUP issues the `dc_clt_send_s` function. To use the message assembly facility or message delivery confirmation facility, the CUP issues the `dc_clt_assem_send_s` function.

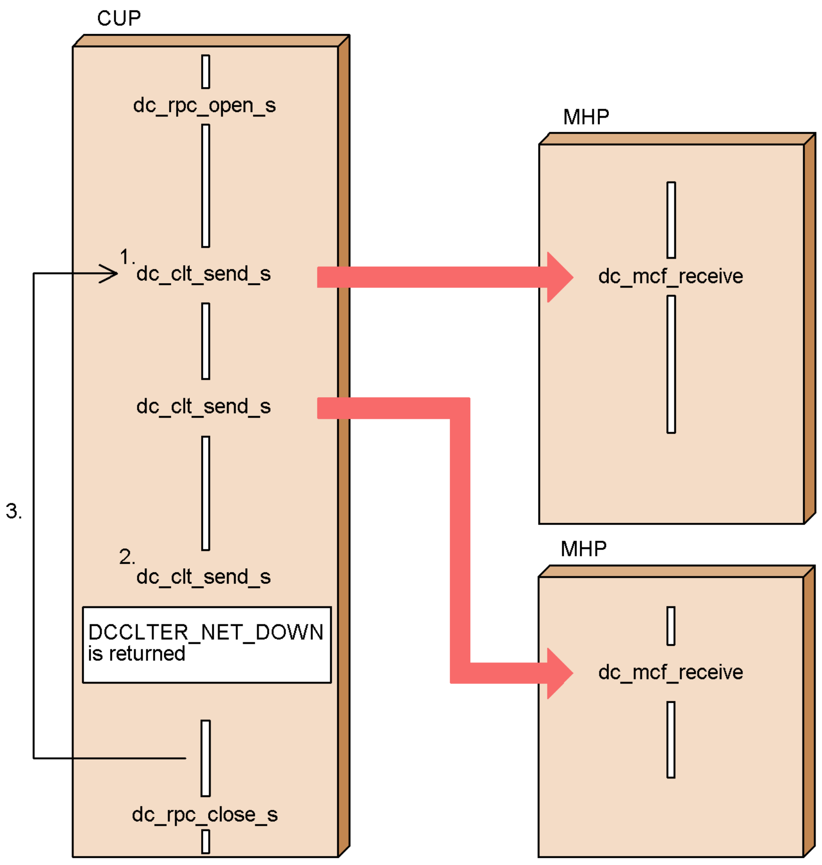
Before sending a send-only message, the following must be specified in the client environment definition.

- Specify the node name of the connection destination in `DCSNDHOST`.
- Specify the port number (port number specified in the `portno` operand of the `mcftalccn` definition command in the MCF communication configuration definition) of the connection destination in `DCSNDPORT`.

It is also necessary to issue the `dc_rpc_open_s` function with `DCCLT_ONEWAY_SND` specified in the flags.

The following figure shows the processing of a send-only message.

Figure 2-18: Processing of send-only messages



Explanation:

1. After the MHP has been started, the CUP is started and issues the dc_clt_send_s function.
2. If the connection is freed from the MHP when the dc_clt_send_s function is issued, the return value DCCLTER_NET_DOWN is returned to the CUP.
3. To send another send-only message, issue the dc_clt_send_s function.

2.5.2 Receive-only messages

A CUP receives messages sent from an MHP. This type of message is called a *receive-only message*.

The CUP issues the `dc_clt_receive_s` function to receive messages from the MHP using the TCP/IP protocol. The CUP issues the `dc_clt_assem_receive_s` function to use the message assembly facility or message delivery confirmation facility.

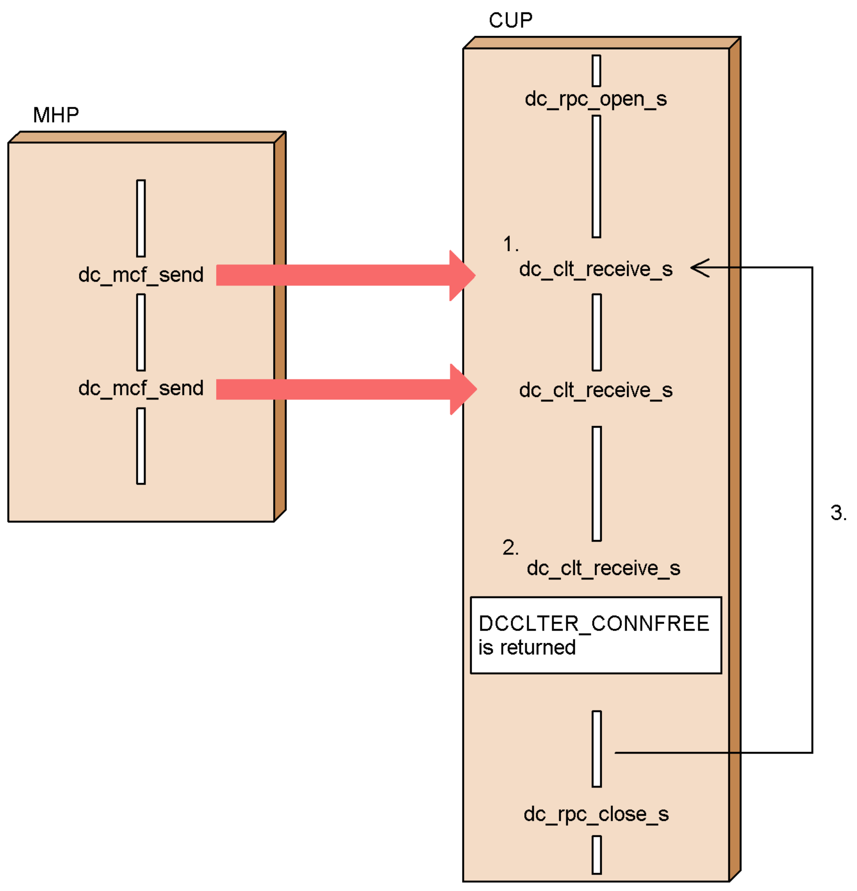
The TCP/IP protocol divides a single message into multiple packages or packs multiple messages into a single packet. TP1/Client determines the end of a received message based on the message length specified by the user. The user must receive the header containing the message length and, on the basis of it, receive the actual message length. Note that the CUP does not need to implement these operations when using the message assembly facility or message delivery confirmation facility.

If the received message is shorter than the specified message length, TP1/Client considers the message is divided. TP1/Client does not return control to the CUP until the specified length of message data has been received. However, if a timeout or error occurs before the specified length of message data can be received and the `dc_clt_receive_s` function is issued, TP1/Client discards the message data that has been received. On the other hand, if the `dc_clt_receive2_s` or `dc_clt_assem_receive_s` function is issued, TP1/Client can return control to the CUP without discarding the message data that has been received so far. Using these functions, you can reserve a received message that does not reach the specified length due to an error. You are responsible for reconstructing incompletely received messages.

Before receive-only messages can be received, the applicable CUP port number (port number specified in the `oportno` operand of the `mcftalccn` definition command in the MCF communication configuration definition) must be specified in `DCRCVPORT` of the client environment definition. It is also necessary to issue the `dc_rpc_open_s` function with `DCCLT_ONEWAY_RCV` specified in the flags.

The following figure shows the processing of receive-only messages.

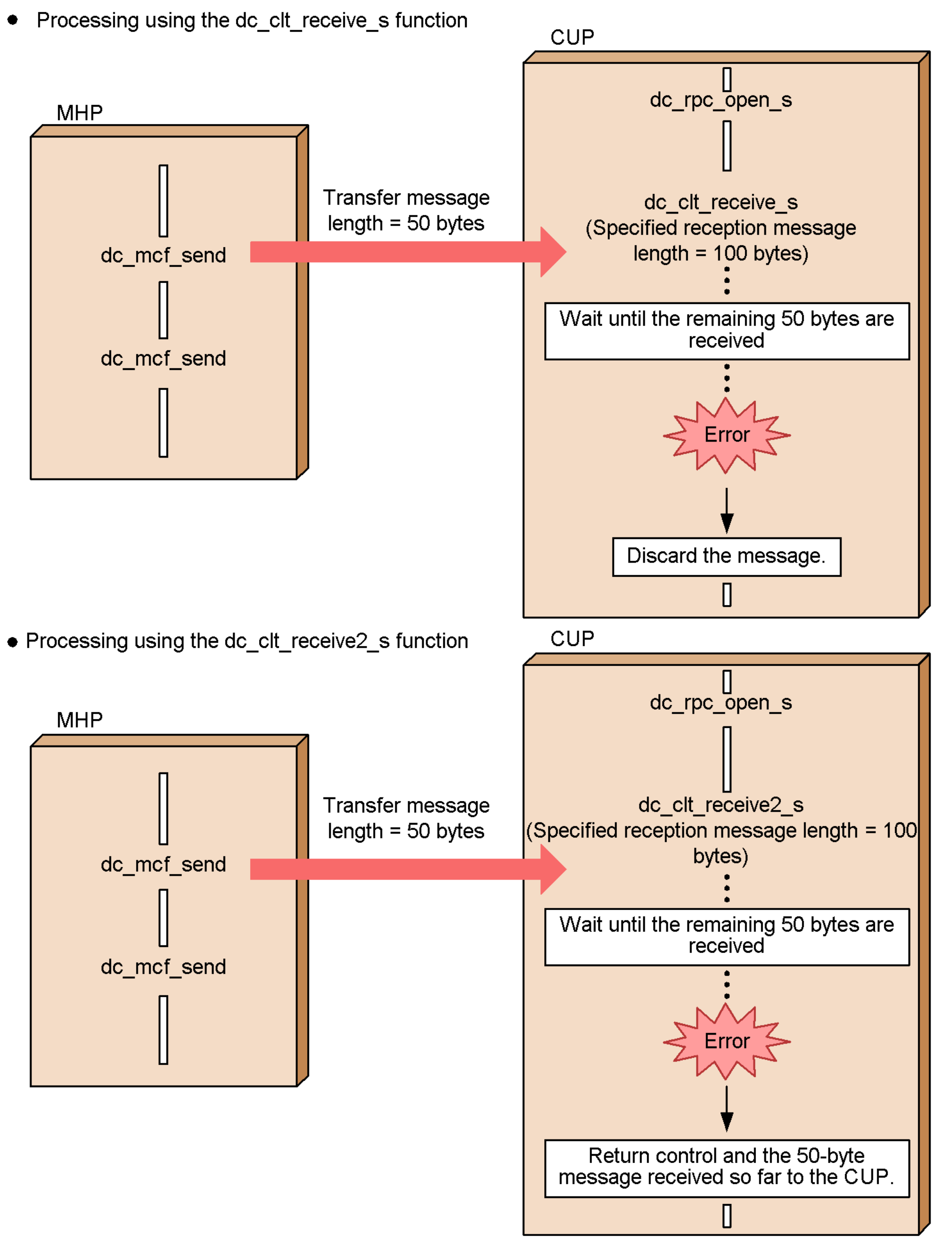
Figure 2-19: Processing of receive-only messages



Explanation:
1. During an MHP retry of a request to establish connection, the CUP starts and issues the dc_clt_receive_s function. If the retry fails, the mcftactcn command is input to establish the connection.
2. When the MHP frees the connection, it returns DCCLTER_CONNFREE to the CUP.
3. To receive another receive-only message, the dc_clt_receive_s function is issued. Note that the mcftactcn command must be input to establish the connection before the function is issued.

The following figure shows the processing of receive-only messages if an error occurs.

Figure 2-20: Processing of receive-only messages if an error occurs


● Processing using the dc_clt_receive_s function
MHP
dc_mcf_send
dc_mcf_send
Transfer message length = 50 bytes
CUP
dc_rpc_open_s
dc_clt_receive_s (Specified reception message length = 100 bytes)
Wait until the remaining 50 bytes are received
Error
Discard the message.
● Processing using the dc_clt_receive2_s function
MHP
dc_mcf_send
dc_mcf_send
Transfer message length = 50 bytes
CUP
dc_rpc_open_s
dc_clt_receive2_s (Specified reception message length = 100 bytes)
Wait until the remaining 50 bytes are received
Error
Return control and the 50-byte message received so far to the CUP.

2.5.3 Sending and receiving messages

Messages can be sent and received between the CUP and MHP. The CUP issues the `dc_clt_send_s` function to send a message to the MHP, and issues the `dc_clt_receive_s` or `dc_clt_receive2_s` function to receive a message from the MHP. When the message assembly facility or message delivery confirmation facility is used, the CUP issues the `dc_clt_assem_send_s` function to send a message to the MHP, and issues the `dc_clt_assem_receive_s` function to receive a message from the MHP.

Before sending or receiving a message, the following must be specified in the client environment definition:

When the MHP type is server:

- Specify the node name of the connection destination in `DCSNDHOST`.
- Specify the port number (port number specified in the `portno` operand of the `mcftalccn` definition command in the MCF communication configuration definition) of the connection destination in `DCSNDPORT`.

When the MHP type is client:

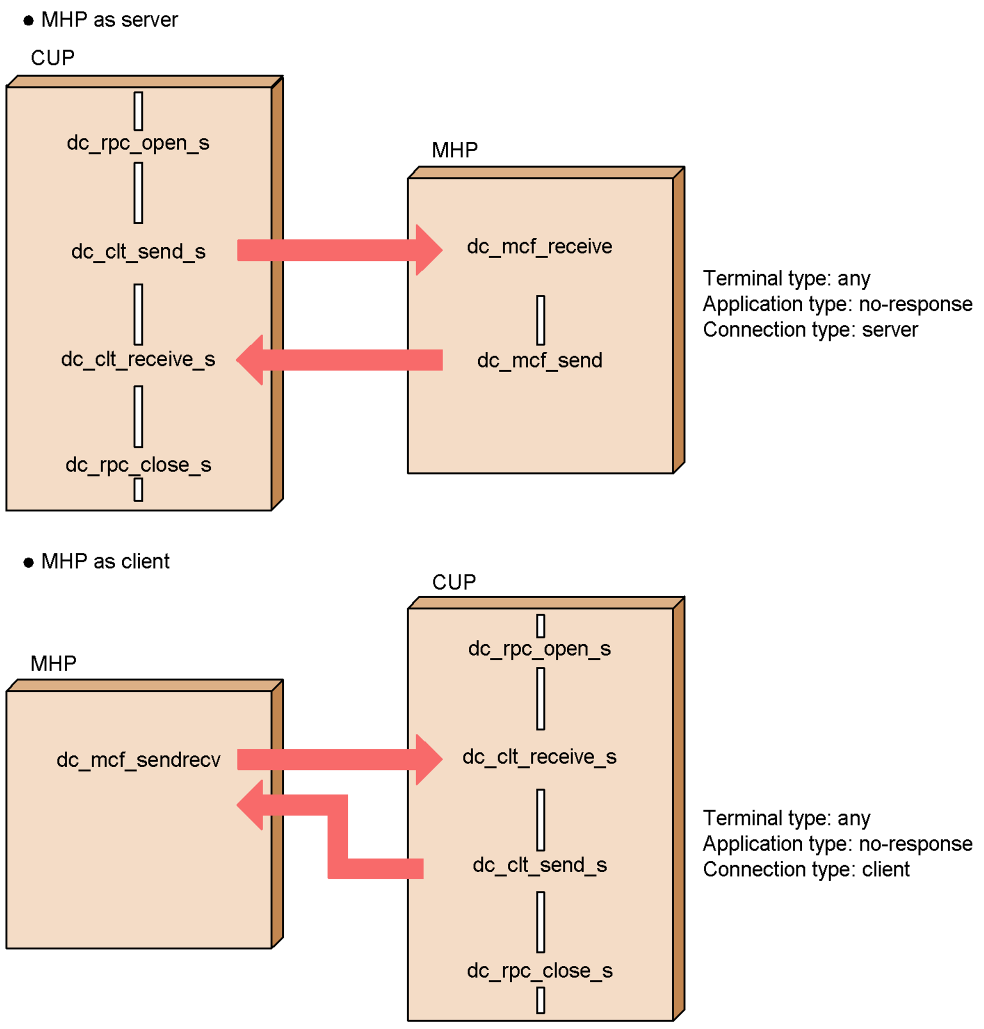
Specify the CUP port number (port number specified in the `oportno` operand of the `mcftalccn` definition command in the MCF communication configuration definition) in `DCRCVPORT`.

It is also necessary to issue the `dc_rpc_open_s` function with `DCCLT_SNDRCV` specified in the flags.

In send-only or receive-only mode, messages are sent or received through independent connections. However, in send and receive mode, all messages are transmitted through the same connection.

The following figure shows the processing of sending and receiving a message.

Figure 2-21: Sending and receiving a message



2.5.4 Message assembly facility and delivery confirmation facility

TP1/Client includes functionality that automatically prefixes four-byte message length information to a message when the message is sent and automatically deletes the information when the message is received. This functionality is called the *message assembly facility*. If you use this facility, you do not need to take the message length into account when creating a CUP.

In addition to the four-byte message length information, you can send a message with one-byte segment information and a six-byte message ID added so that response-only data can be received to confirm delivery of the message. This functionality is called the *message delivery confirmation facility*. If you use this facility, you do not need to take into account the validity of a message and whether a message that has been sent is delivered when you create a CUP.

The message assembly facility and message delivery confirmation facility are implemented by using the following functions to send and receive messages:

- `dc_clt_assem_send_s` function
- `dc_clt_assem_receive_s` function

Whether the message assembly facility or message delivery confirmation facility is used when these functions are issued is specified by using `DCCLTDELIVERYCHECK` of the client environment definition as shown in the following table.

Table 2-4: Relationship between the facility to be used and the DCCLTDELIVERYCHECK specification of the client environment definition

Facility to be used	DCCLTDELIVERYCHECK specification of the client environment definition
Message assembly facility	Specify `N` or omit `DCCLTDELIVERYCHECK`.
Message delivery confirmation facility	Specify `Y`.

(1) Conditions for using the facilities

Both facilities require issuing of the `dc_rpc_open_s` function with `DCCLT_ONEWAY_SND`, `DCCLT_ONEWAY_RCV`, or `DCCLT_SNDRCV` specified in the `flags` argument.

In addition, TP1/NET/TCP/IP in the remote system must be set to use the message assembly facility or message delivery confirmation facility. For details about how to set TP1/NET/TCP/IP, see the manual *OpenTP1 Version 7 Protocol TP1/NET/TCP/IP*. Note, however, that the remote system need not use TP1/NET/TCP/IP if send messages and receive messages have the same format.

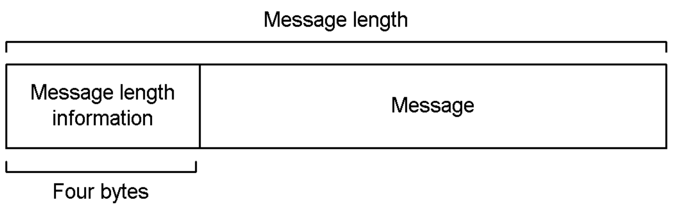
(2) Message formats

The following explains the format of messages sent or received when the message assembly facility and the message delivery confirmation facility are used.

(a) Format of messages sent or received when the message assembly facility is used

The following figure shows the format of messages sent or received when the message assembly facility is used.

Figure 2-22: Format of messages sent or received when the message assembly facility is used



Note that the message length indicates the length of the assembled message.

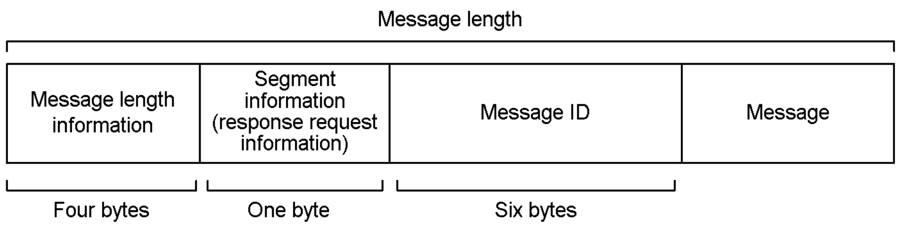
(b) Format of messages sent or received when the message delivery confirmation facility is used

This subsection explains the format of messages sent or received when the message delivery confirmation facility is used, and the format of response-only data.

- Format of messages sent or received

  The following figure shows the format of messages sent or received between TP1/ Client and TP1/NET/TCP/IP.

  Figure 2-23: Format of messages sent or received when the message delivery confirmation facility is used


Message length
Message length information
Segment information (response request information)
Message ID
Message
Four bytes
One byte
Six bytes

  The following explains the items in the above figure.

  - Message length information

    This field contains the length of the assembled message.

  - Segment information

    This field contains information that indicates the segment type and a response request. The value set here is `0x18`, which indicates a single segment and an automatic-response request.

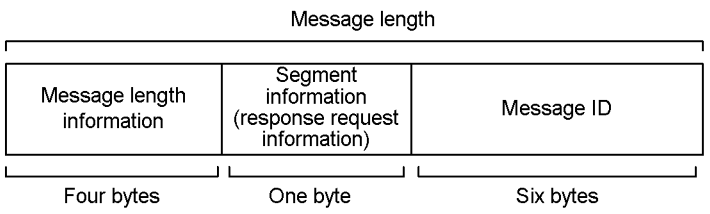
  - Message ID

    This field contains information that is used to check the combination of message and response-only data. A unique value is set in this field each time a message is sent.

- Format of response-only data

The following figure shows the format of response-only data sent or received when the message delivery confirmation facility is used.

Figure 2-24: Format of response-only data


Message length
Message length information
Segment information (response request information)
Message ID
Four bytes
One byte
Six bytes

The following explains the items in the above figure.

- Message length information

  This field contains the length of the assembled message.

- Segment information

  This field contains information that indicates the segment type and a response request. The value set here is `0x10`, which indicates a single segment and response-only data.

- Message ID

  This field contains information that is used to check the combination of message and response-only data. The message ID added to the received message is set in this field. TP1/Client compares the message ID that it added when it sent the message with the message ID that TP1/NET/TCP/IP added to the response-only data. If these message IDs match, TP1/Client closes the connection.

(3) Flow of sending and receiving a message when the message assembly facility is used

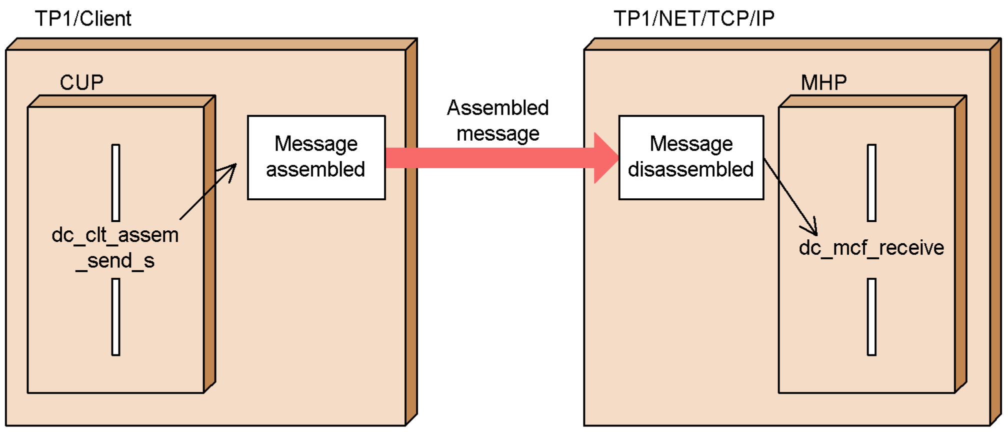
This subsection explains how a message is assembled and disassembled when the message assembly facility is used.

(a) When a message is sent

When TP1/Client uses the message assembly facility to send a message, the `dc_clt_assem_send_s` function is issued. Four-byte message length information is prefixed to the message.

The following figure shows the flow of sending a message when the message assembly facility is used.

Figure 2-25: Flow of sending a message when the message assembly facility is used



If you want to release a connection immediately after a message is sent, specify `DCCLT_SND_CLOSE` in the `flags` argument of the `dc_clt_assem_send_s` function. If you specify `DCNOFLAGS`, the connection is not released until the `dc_rpc_close_s` function is issued (except when an error has occurred).

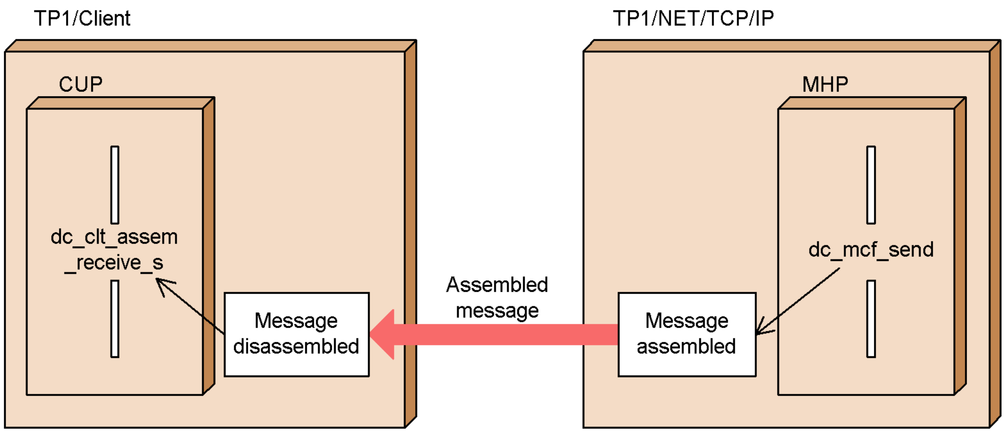
The host name of TP1/NET/TCP/IP with which TP1/Client communicates is specified in `DCSNDHOST` of the client environment definition. The port number is specified in `DCSNDPORT` of the client environment definition. You can also specify the host name and port number in arguments of the `dc_clt_assem_send_s` function.

(b) When a message is received

When TP1/Client uses the message assembly facility to receive a message, the `dc_clt_assem_receive_s` function is issued. TP1/Client receives message data for the message length prefixed to the message body when the message was sent.

The following figure shows the flow of receiving a message when the message assembly facility is used.

Figure 2-26: Flow of receiving a message when the message assembly facility is used



If you want to release a connection immediately after a message is received, specify `DCCLT_RCV_CLOSE` in the `flags` argument of the `dc_clt_assem_receive_s` function. If you specify `DCNOFLAGS`, the connection is not released until the `dc_rpc_close_s` function is issued (except when an error has occurred).

The CUP port number is specified in `DCRCVPORT` of the client environment definition.

(4) Flow of sending and receiving a message when the message delivery confirmation facility is used

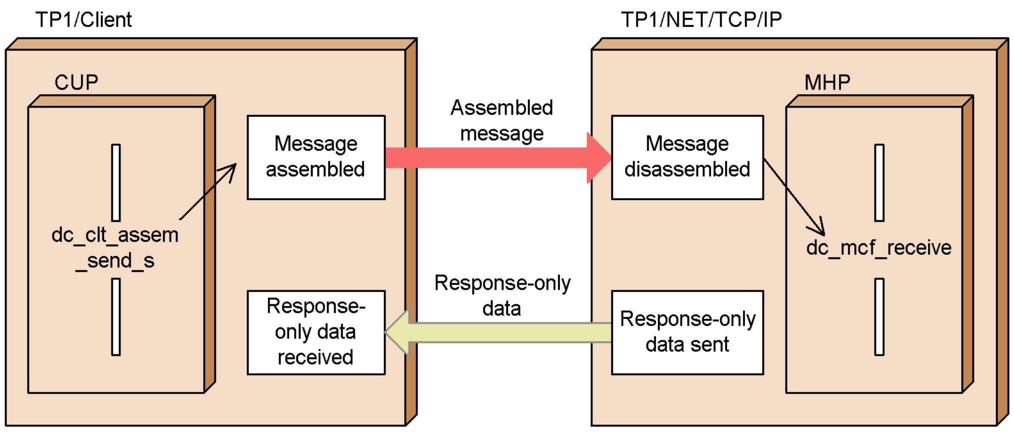
This subsection explains how a message is assembled and disassembled, and the flow of sending and receiving response-only data when the message delivery confirmation facility is used.

(a) Sending a message and confirming delivery

When TP1/Client uses the message delivery confirmation facility to send a message, the `dc_clt_assem_send_s` function is issued. After sending the message, TP1/Client waits for response-only data from TP1/NET/TCP/IP. When it receives the response-only data, TP1/Client returns control to the CUP. The received response-only data is not reported to the CUP.

The following figure shows the flow of sending a message when the message delivery confirmation facility is used.

Figure 2-27: Flow of sending a message when the message delivery confirmation facility is used



When the message delivery confirmation facility is used, you can monitor the sequence of sending a message from the time the message is sent until response-only data is received. If the monitoring times out, TP1/Client releases the connection with TP1/NET/TCP/IP, and returns `DCCLTER_TIMED_OUT`.

TP1/Client receives response-only data over the same connection it used to send the message. If you want the connection to be released as soon as the response-only data is received, specify `DCCLT_SND_CLOSE` in the `flags` argument of the `dc_clt_assem_send_s` function. If you specify `DCNOFLAGS`, the connection is not released until the `dc_rpc_close_s` function is issued (except when an error has occurred).

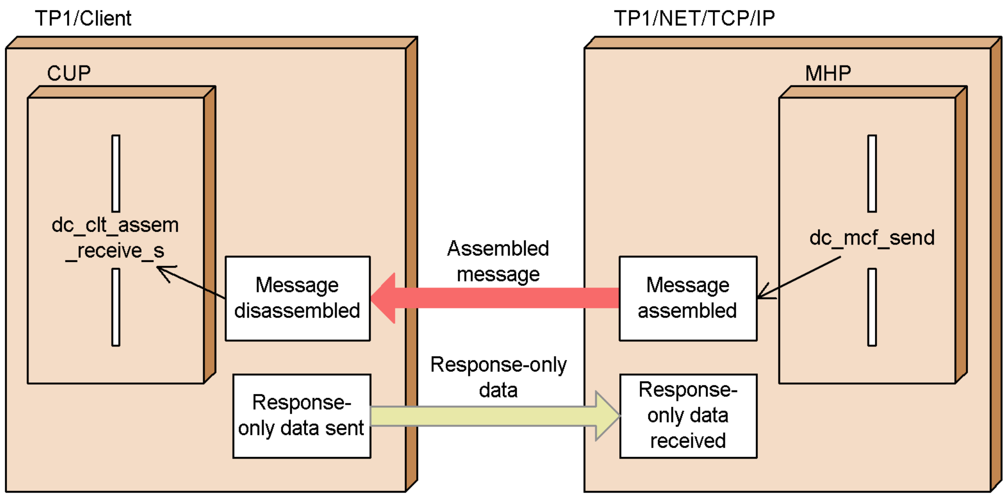
The host name of TP1/NET/TCP/IP with which TP1/Client communicates is specified in `DCSNDHOST` of the client environment definition. The port number is specified in `DCSNDPORT` of the client environment definition. You can also specify the host name and port number in arguments of the `dc_clt_assem_send_s` function.

(b) Receiving a message and confirming delivery

When TP1/Client uses the message delivery confirmation facility to receive a message, the `dc_clt_assem_receive_s` function is issued. After receiving the message, TP1/Client sends response-only data to TP1/NET/TCP/IP and returns control to the CUP.

The following figure shows the flow of receiving a message when the message delivery confirmation facility is used.

Figure 2-28: Flow of receiving a message when the message delivery confirmation facility is used



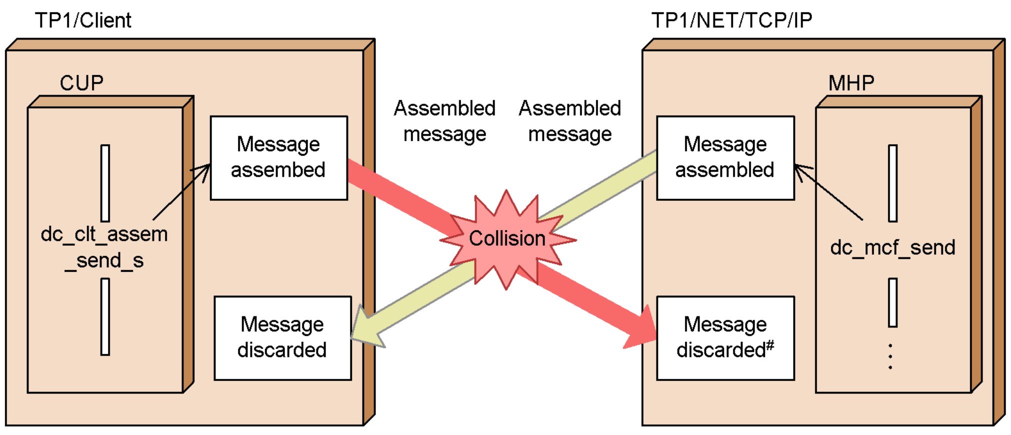
TP1/Client sends response-only data over the same connection that it used to receive the message. If you want the connection to be released as soon as response-only data is sent, specify `DCCLT_RCV_CLOSE` in the `flags` argument of the `dc_clt_assem_receive_s` function. If you specify `DCNOFLAGS`, the connection is not released until the `dc_rpc_close_s` function is issued (except when an error has occurred).

(c) Collision between send and receive messages

If a message from TP1/Client collides with a message from TP1/NET/TCP/IP, TP1/Client discards the received message, releases the connection with TP1/NET/TCP/IP, and then returns `DCCLTER_COLLISION_MESSAGE`.

The following figure shows the flow of processing when send and receive messages collide.

Figure 2-29: Flow of processing when send and receive messages collide



#: How TP1/NET/TCP/IP operates following a collision depends on the settings.

(5) Validity checking

When the message assembly facility or message delivery confirmation facility is used, TP1/Client automatically checks the validity of sent and received messages.

(a) Validity check for message length (message assembly facility)

When the message assembly facility is used and TP1/Client receives a message, TP1/Client checks the validity of the message length. The following table shows the handling if an error is found.

Table 2-5: Handling of an error detected by the validity check for message length

No.	Error	System	User response
1	An invalid message is received (the message length is 0 to 4 bytes).	Stops processing.	Review the remote system settings.
2	The receive buffer is insufficient (the message length minus four bytes is longer than the value of the `recvleng` argument).	Stops processing.	Check whether a valid value is specified in the `recvleng` argument of the `dc_clt_assem_receive_s` function. Alternatively, review the MHP.

(b) Validity check for response-only data (message delivery confirmation facility)

When the message delivery confirmation facility is used and TP1/Client sends a message, TP1/Client receives response-only data and checks the validity of the data. The following table shows the handling of an error if detected.

Table 2-6: Handling of an error detected by the validity check for response-only data

No.	Error	System	User response
1	Any of the following invalid messages is received: • Message whose length is 0 to 10 bytes • Message whose length is 11 bytes and whose segment information is not `0x10` • Message whose segment information is not `0x10` or `0x18`	Stops processing.	Review the remote system settings.
2	Messages collide.	Stops processing.	Try again, if necessary.
3	Message IDs do not match.	Retries reception.	None

(c) Validity check for a received message (message delivery confirmation facility)

When the message delivery confirmation facility is used and TP1/Client receives a message, TP1/Client checks the validity of the message. If TP1/Client finds an error, it stops processing. The following table shows the handling of an error if detected.

Table 2-7: Handling of an error detected by the validity check for a received message

No.	Error	System	User response
1	Any of the following invalid messages is received: • Message whose length is 0 to 11 bytes • Message whose segment information is not `0x10` or `0x18`	Stops processing.	Review the remote system settings.
2	The receive buffer is insufficient (the message length minus 11 bytes is longer than the value of the `recvleng` argument).	Stops processing.	Check whether a valid value is specified in the `recvleng` argument of the `dc_clt_assem_receive_s` function. Alternatively, review the MHP.

2.5.5 Notes on using the TCP/IP communication facility

This subsection provides notes on using the TCP/IP communication facility.

(1) Sending messages

(a) Loss of a message in the event of an error

If an error event as described below occurs, the TP1/Client cannot detect a message loss. The user should assign serial numbers to messages so that any message loss can be detected. However, if the message delivery confirmation facility is used, processing in the CUP that assigns serial numbers is not required because TP1/Client manages message serial numbers.

- A communication error occurs or the connection is freed immediately after the message sent from TP1/Client is written in the buffer of the socket and is normally terminated.
- A communication error occurs or the connection is freed immediately before the message sent from TP1/Client is written in the receive buffer of the MHP.

(b) Establishing a connection

TP1/Client acts as a client and sends a message to an MHP. TP1/Client establishes a connection to the MHP. When the MHP uses TP1/NET/TCP/IP, a server type connection is established.

(2) Receiving messages

(a) Loss of a message in the event of an error

If an error event as described below occurs, the TP1/Client cannot detect a message loss. The user should assign serial numbers to messages so that any message loss can be detected. However, if the message delivery confirmation facility is used, processing in the CUP that assigns serial numbers is not required because TP1/Client manages message serial numbers.

- A communication error occurs or the connection is freed immediately after the message sent from the MHP is written in the buffer of the socket and is normally terminated.
- A communication error occurs or the connection is freed immediately before the message sent from the MHP is written in the receive buffer of TP1/Client.

(b) Checking received messages

A message from any MHP can be received. When receiving a request to establish a connection, a CUP accepts it unconditionally and receives a message. By including a header with the message identifier in each message, the user must check whether the message is to be received by the CUP.

(c) Message length

Messages are received using the TCP/IP protocol.

The TCP/IP protocol divides a single message into multiple packages or packs multiple messages into a single packet. TP1/Client determines the end of a received message based on the message length specified by the user. The user must receive the header containing the message length and, on the basis of it, receive the actual message length. However, if the message assembly facility or message delivery confirmation facility is used, this processing need not be implemented in the CUP because TP1/Client manages the message length.

If the received message is shorter than the specified message length, TP1/Client considers the message is divided. Therefore, it does not return control to the CUP until the message of the specified length is received.

(d) Establishing a connection

TP1/Client acts as a server and receives a message from an MHP.

The MHP establishes a connection to TP1/Client. When the MHP uses TP1/NET/TCP/IP, a client type connection is established.

(3) Other notes

The following provides notes on using the message assembly facility and the message delivery confirmation facility in TP1/NET/TCP/IP. If you use the message assembly facility and message delivery confirmation facility in TP1/Client, you can skip these notes.

For details about the protocol-specific definitions for TP1/NET/TCP/IP, see the manual *OpenTP1 Protocol TP1/NET/TCP/IP*.

(a) Message assembly facility in TP1/NET/TCP/IP

In TP1/NET/TCP/IP, when you use the message assembly facility, a four-byte message length is added to the beginning of the data sent by an MHP. When you send data to an MHP, you must specify the message length in the first four bytes. The message length must be in the network byte order. The message length is deleted when the MHP receives the message. Note that you must mind the message length when sending or receiving messages to or from CUPs.

(b) Message delivery confirmation facility in TP1/NET/TCP/IP

When the message delivery confirmation facility is used in TP1/NET/TCP/IP, 11-byte message information is prefixed to data sent by the MHP. When TP1/Client receives the data, it sends response-only data. (Note that 11-byte message information must also be prefixed to the data to be sent to the MHP.) After the MHP sends the data, it waits to receive the response-only data.

Ensure that the network byte order is used for the message length information in the message information prefixed to the send data. When the MHP receives a message, the message length information is deleted. However, the CUP must set the message information while sending and receiving data.

2.6 Facility for receiving one-way messages from the server

This section describes the facility for receiving one-way messages (messages sent from the server to clients).

2.6.1 Overview of the facility for receiving one-way messages from the server

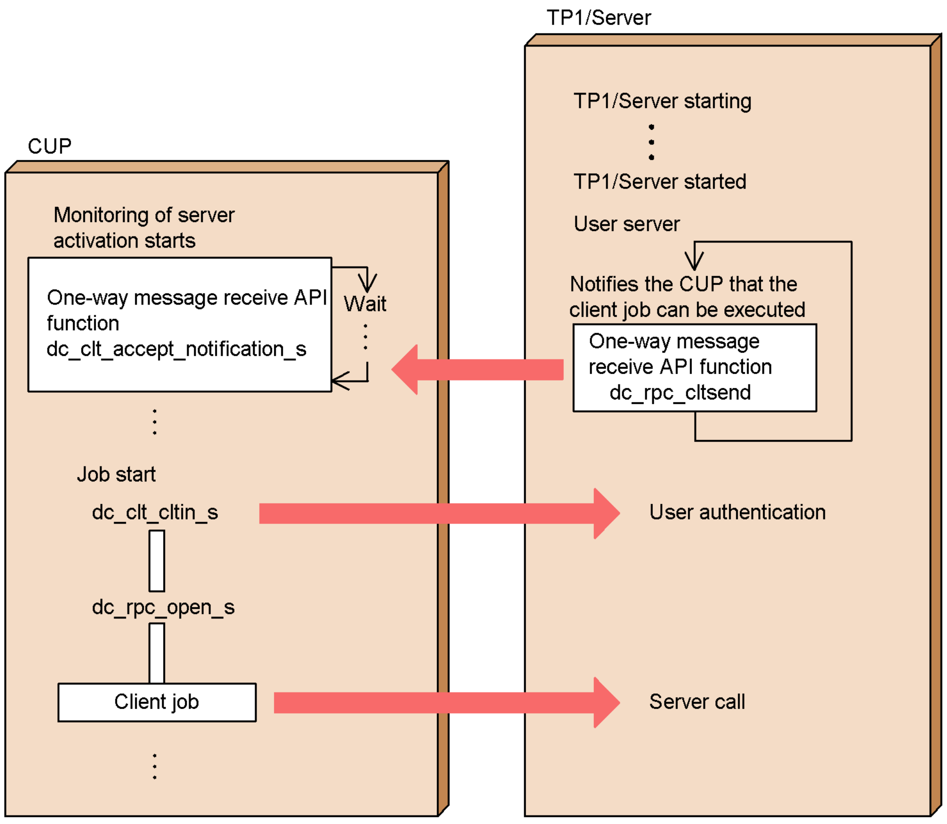
The *facility for receiving one-way messages from the server* can be used to deliver online start signals concurrently to the client. This is analogous to OLTP of a mainframe starting all terminals at one time.

To use the facility for receiving one-way messages from the server, issue the `dc_clt_accept_notification_s` function. This function allows the client to wait for a message from the server within the time specified for API regardless of the server state whether it is active or inactive.

When the server sends a message at startup, the client detects the server startup and is ready for starting a user job (CUP). The following figure illustrates the one-way message reception function.

The following figure shows the processing flow for the facility for receiving one-way messages from the server.

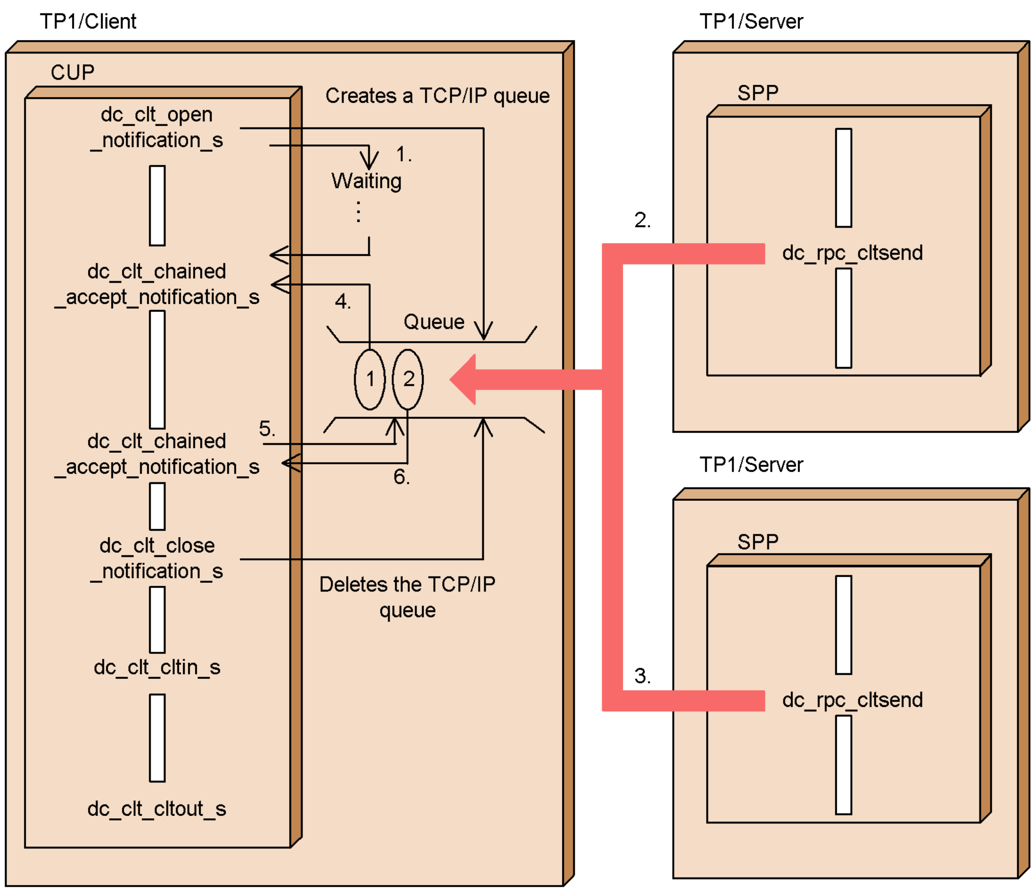
Figure 2-30: Processing flow for facility for receiving one-way messages from the server


TP1/Server
TP1/Server starting
TP1/Server started
User server
Notifies the CUP that the client job can be executed
One-way message receive API function dc_rpc_cltsend
User authentication
Server call
CUP
Monitoring of server activation starts
One-way message receive API function dc_clt_accept_notification_s
Wait
Job start
dc_clt_cltin_s
dc_rpc_open_s
Client job

2.6.2 Overview of the continuous reception function for one-way messages

The *continuous reception function for one-way messages* allows the client to receive continuous one-way messages from the server. This function starts when the `dc_clt_open_notification_s` function is executed and terminates when the `dc_clt_close_notification_s` function is executed. Ordinarily, if the server sends a one-way message to the client when the client is not ready to receive one-way messages from the server, an error is returned. However, if the continuous reception function for one-way messages is used, no error is returned because the messages are stored in the TCP/IP queue. The messages leave the queue when the client issues the function that receives one-way messages. The following figure shows the processing flow for the continuous reception function for one-way messages.

Figure 2-31: Processing flow for the continuous reception function for one-way messages



Explanation:
1. TP1/Client is waiting for a one-way message.
2. and 3.
   TP1/Server is started, and uses a one-way message to notify the CUP that the client is ready to execute jobs. The one-way messages from the server are stored in the TCP/IP queue.
4. TP1/Client retrieves a one-way message from the TCP/IP queue, and returns control to the CUP.
5. and 6.
   When the dc_clt_chained_accept_notification_s function is issued, a one-way message from the server has arrived in the TCP/IP queue. Accordingly, TP1/Client retrieves this message, and returns control to the CUP.

2.6.3 Notes on using the continuous reception function for one-way messages

The following provides notes on using the continuous reception function for one-way messages:

- In the client environment definition, if `DCSELINT` is set to 0, control is not returned to the operating system.
- The maximum number of messages that the TCP/IP queue can contain depends on the maximum defined in the operating system. If more messages than the maximum arrive at the queue, the `dc_rpc_cltsend` function executed on the server side returns a `DCRPCER_SERVICE_NOT_UP` error.
- With TP1/Client/P, a problem exists such that even if you specify an already used port number, an error will not be detected. Since TP1/Client/P does not check for duplicated port number specifications, make sure that you do not specify the number of an already used port.

2.7 XATMI interface facility

An RPC can send and receive a limited length of data. Large data such as images may exceed the specified RPC data length. TP1/Client uses the interactive service, an XATMI interface facility, for sending and receiving data.

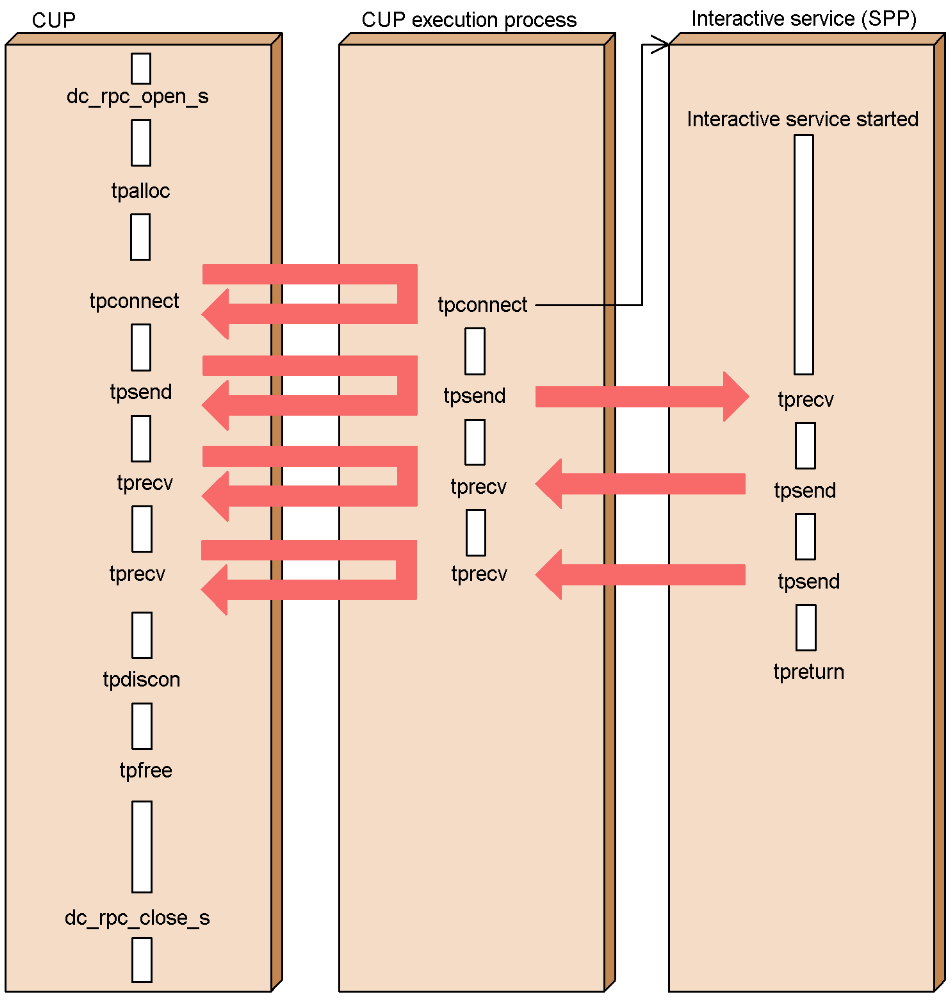
Communication by the interactive service divides large data into units called packets. Thus, the complete data is sent by sending these packets.

To start the interactive service communication, issue the `tpalloc` function to allocate typed buffer. When the buffer is allocated, issue the `tpconnect` function to establish connection the same way as for normal message exchange. Issue the `tprecv` function and `tpsend` function of the XATMI interface facility to send and receive messages. To terminate message exchange, issue the `tpdiscon` function to disconnect the connection, and then issue the `tpfree` function to release the typed buffer. A return value is sent if an error occurred during interactive service.

The interactive service communication is the only XATMI interface facility available for TP1/Client.

The following figure shows the process flow of communication by the XATMI interface facility interactive service.

Figure 2-32: Process flow of communication by interactive service



Note: A tpxxxx function has no suffix "_s."

2.7.1 Interactive service

(1) Establishing connection

The CUP establishes connection with the interactive service by issuing the `tpconnect` function. The UAP process that has established connection by the `tpconnect` function is called an *originator*. The other UAP process at the end of connection is called a *subordinator*.

The descriptor that identifies the established connection is returned upon normal termination of the `tpconnect` function. Set this descriptor for each of the functions used during the communication.

Issuing the `tpreturn` function to terminate processing under the interactive service disconnects the connection.

The flags argument of the `tpconnect` function allows you to specify whether to give control. A process specified to have control can issue the `tpsend` function to send data. The process specified to have no control passes control to the remote process in communication. The process that calls the `tpconnect` function can issue the `tprecv` function to receive data.

(2) Sending data

Issue the `tpsend` function for sending data. To issue this function, the descriptor returned by the `tpconnect` function must be set in the argument to specify which connection is to be used.

Only the process that has control of connection can issue the `tpsend` function. The `tpsend` function issued by the process that has no control will return an error.

If you want to pass control of connection to the remote process, specify the `tpsend` function argument.

(3) Receiving data

Issue the `tprecv` function for receiving data. Data is received asynchronously. Only a process that has no control of connection can issue the `tprecv` function.

You can specify the argument to have the `tprecv` function wait for receiving data. When the process that calls the `tprecv` function is under a transaction, the maximum waiting time is the value specified in the `DCCLTTREXPTM` of the client environment definition or in the `trn_expiration_time` operand of the client service definition. In this case, the CUP execution process terminates abnormally when the maximum waiting time is expired; the `tprecv` function does not return an error.

When the process that calls the `tprecv` function is not under a transaction, the maximum waiting time is the value specified in the `watch_time` operand of the client service definition. In this case, the `tprecv` function returns an error when the maximum waiting time is expired.

(4) Disconnecting the connection

After termination of the interactive service, issuing the `tpreturn` function disconnects the connection normally. Connection may also be disconnected if a communication error occurred.

(5) Forcibly disconnecting the connection

Issue the `tpdiscon` function for forcibly disconnecting the connection. The descriptor set in the `tpdiscon` function is no longer effective in processing. The transaction is rolled back.

(6) Generating a transaction

Issue the `dc_trn_begin_s` function for generating a transaction by the CUP that uses the XATMI interface for communication.

Table 2-4 shows when a transaction is generated.

Table 2-8: Transaction generation time

The dc_clt_connect_s function is issued.		The dc_clt_connect_s function is not issued.	
The tpconnect function is issued.	**The tpconnect function is not issued.**	**The tpconnect function is issued.**	**The tpconnect function is not issued.**
Y	Y	Y	No

Legend:

Y: A transaction can be generated and communication with the interactive service via the XATMI interface is available.

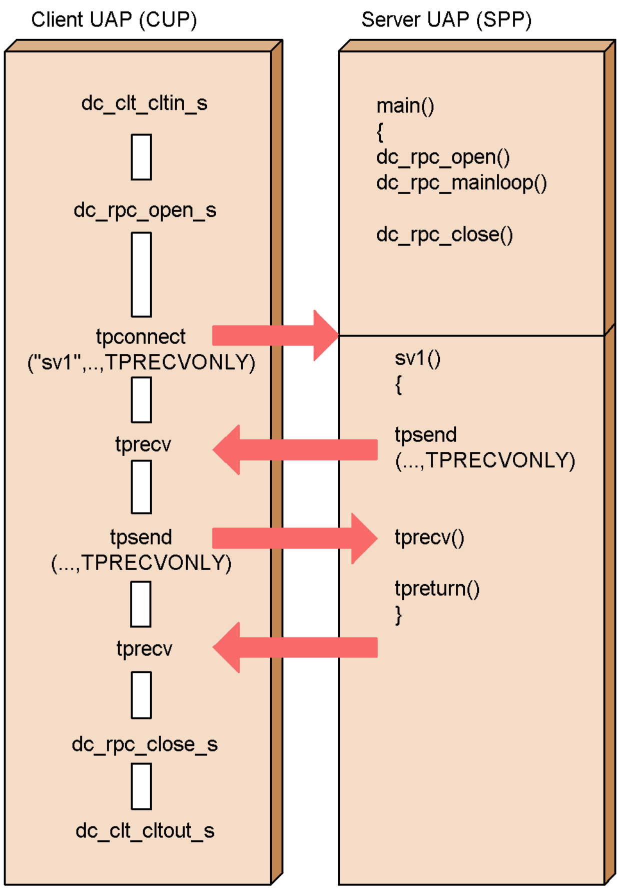
No: A transaction can be generated, but communication with the interactive service via the XATMI interface is unavailable.

(7) Operating with the OpenTP1 dc_rpc_call_s function

The `dc_rpc_call_s` function can still be issued after the `tpconnect` function has established connection with the interactive service.

The following figure shows the communication mode of the interactive service.

Figure 2-33: Communication mode of interactive service



Note: A tpxxxx function has no suffix "_s."

2.7.2 Interactive service time monitoring

Any time monitoring for interactive service communication must follow the value specified in the OpenTP1 definitions.

A timeout error of interactive service occurs when any of the following is expired:

- Maximum response waiting time
- Maximum time interval in permanent connection
- Transaction branch expiration time

(1) Timeout of maximum response waiting time

The CUP and the CUP execution process detect expiration of the maximum response waiting time. For the CUP, specify the value for monitoring in `DCWATCHTIM` of the client environment definition. For the CUP execution process, specify the `watch_time` operand of the client service definition.

The value of `DCWATCHTIM` in the client environment definition must be greater than the value of the `watch_time` operand of the client service definition to keep consistency of the system. If the CUP detects a timeout error earlier than the CUP execution process, communication is disabled until the `dc_rpc_close_s` function is issued. In this case, the CUP execution process forcibly rolls back the transaction (only when the transaction is active) after the timeout of maximum time interval in inquiry response, and then disconnects the connection with the CUP.

(2) Timeout of maximum time interval in permanent connection

The CUP execution process detects expiration of the maximum time interval in permanent connection. Specify the value for monitoring in the `DCCLTINQUIRETIME` of the client environment definition, or in the `clt_inquire_time` of the client service definition. The priority of the definitions is as follows:

```
Client environment definition > Client service definition
```

The maximum time interval in permanent connection means the maximum time interval between inquiries (including the `tpsend` function) from the CUP to the CUP execution process.

The CUP execution process that detects a timeout error forcibly rolls back the transaction (only when the transaction is active), and then disconnects the connection with the CUP.

(3) Timeout of transaction branch expiration time

The CUP execution process detects the transaction branch expiration time. Specify the value for monitoring in the `DCCLTTREXPTM` of the client environment definition, or in the `trn_expiration_time` of the client service definition. The priority of the definitions is as follows:

```
Client environment definition > Client service definition
```

The transaction branch expiration time means the time between generation of a transaction and synchronous point acquisition.

The CUP execution process that detects a timeout error terminates abnormally. In this case, all connections are forcibly disconnected and the transaction is rolled back.

You can specify to have the system wait infinitely without detecting timeout except for the transaction branch expiration time. The timeout error due to expiration of the transaction branch expiration time will occur regardless of specification in the definitions.

2.7.3 Receiving events

When an event is set in the descriptor that identifies the connection, that event can be received using the `tpsend` and `tprecv` functions of the interactive service. Information about data exchange is set in the event.

See Section *3.1 Function interface* for details on events.

2.7.4 Communication data type

The data type available for the TP1/Client interactive service is `X_OCTET`.

`X_OCTET` means a byte (character) array whose contents and operation are completely defined by the application. Therefore, the parameter that indicates the data length must be specified for `X_OCTET` to make it clear the length of the character array to be sent by communication management. The specified data length is provided as the `len` parameter for input of the interactive service, and is collected as the `len` parameter for output. The typed buffer specified here does not encode or decode data even when data is exchanged between machines of different models. `X-OCTEX` used for TP1/Client has no subtype.

2.7.5 Notes on using the XATMI interface facility

The following shows notes on using the XATMI interface for TP1/Client.

- Be sure to specify the following values in the definitions when using the XATMI interface under transactions:

  Specify a value other than zero for the `DCCLTTREXPTM` or `trn_expiration_time`.

  Specify `Y` for the `DCCLTTREXPSP` or `trn_expiration_time_suspend`.
- If the blocking status that occurred during data transmission by the `tpconnect` or `tpsend` function is not canceled after a certain period of time, `TPESYSTEM` is returned regardless of whether blocking is specified.
- If the timeout error occurred for the transaction branch expiration time, the CUP execution process terminates abnormally without returning `TPETIME`. In this case, all the connections established before the timeout error are disconnected and no longer available.
- If a timeout error occurred for the maximum time interval in permanent connection, all the connections established before the timeout error are disconnected and no longer available.

- If a communication error occurred in the connection between the CUP and the CUP execution process, the CUP execution process terminates abnormally. All the connections established before the timeout error are disconnected and no longer available.
- Specify `DCCLTXATMI=Y` in the client environment definition.

2.8 Character code converter

This feature is only available for TP1/Client/P.

When using the mainframe computer as the server, the server and the client may use different character code systems. In that case, code conversion must be performed before RPCs. A code can be converted with or without using a code mapping table.

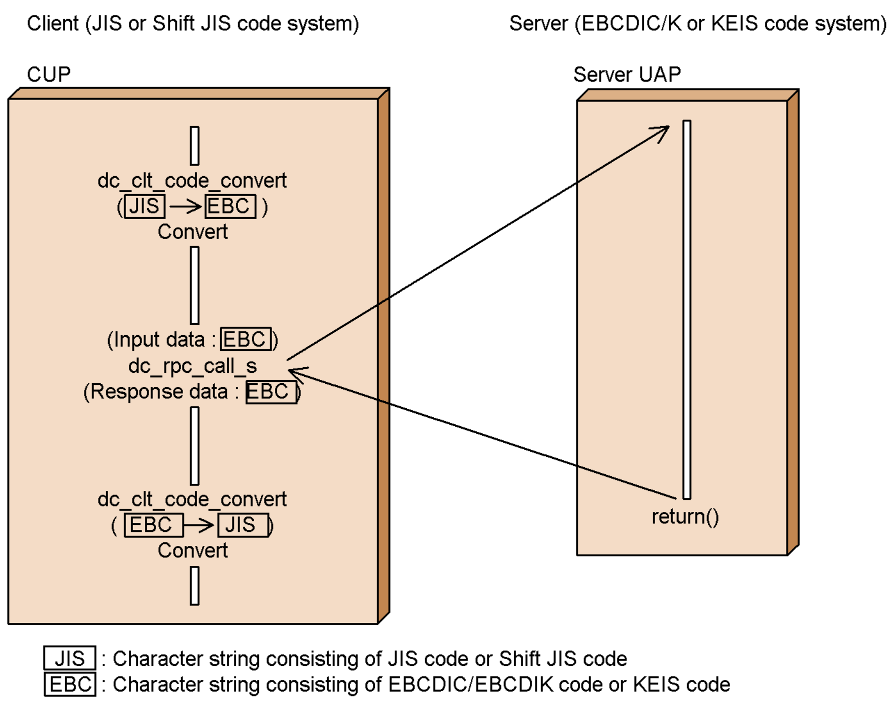
2.8.1 When not using a code mapping table

When TP1/Client/P (Windows environment) issues the `dc_clt_code_convert` function by specifying `DCCLT_JISSJIS_TO_EBCKEIS` for the request code, the character strings consisting of JIS code or Shift JIS code can be converted to the character strings consisting of EBCDIC/EBCDIK code or KEIS code. The client passes the converted character strings to the server to make requests using the `dc_rpc_call_s` function.

When the `dc_clt_code_convert` function is issued with `DCCLT_EBCKEIS_TO_JISSJIS` for the request code to the response message after the `dc_rpc_call_s` function is executed, the character strings consisting of EBCDIC/EBCDIK code or KEIS code are converted to character strings consisting of JIS code or Shift JIS code.

The following figure gives an overview of the character code converter.

Figure 2-34: Overview of the character code converter


Client (JIS or Shift JIS code system)
Server (EBCDIC/K or KEIS code system)
CUP
Server UAP
dc_clt_code_convert
(JIS → EBC)
Convert
(Input data : EBC)
dc_rpc_call_s
(Response data : EBC)
dc_clt_code_convert
(EBC → JIS)
Convert
return()
JIS : Character string consisting of JIS code or Shift JIS code
EBC : Character string consisting of EBCDIC/EBCDIK code or KEIS code

2.8.2 When using a code mapping table

Code conversion using a code mapping table links to CommuniNet to convert gaiji and other codes into specific code character strings. Those codes are converted in the same way as performing conversion without using a code mapping table.

If a code mapping table does not contain a code to be converted, processing is performed in the same way as performing conversion without a code mapping table. An illegal code detected is considered as an error.

(1) Scope

The character code converter can convert character codes covered in CommuniNet.

(2) Usage

When using the converter, issue and call functions in the following order.

1. Start of character code conversion

   `dc_clt_codeconv_open()`

2. Execution of character code conversion (Conversion can be executed two or more times)

`dc_clt_codeconv_exec()`

3. Termination of character code conversion

   `dc_clt_codeconv_close()`

Once character code conversion is started (`dc_clt_codeconv_open()`), you can execute character code conversion (`dc_clt_codeconv_exec()`) two or more times until you terminate the conversion (`dc_clt_codeconv_close()`).

(3) Notes on using a code mapping table

- The use of this converter requires a CommuniNet code mapping table. Use the CommuniNet code mapping utility to create a code mapping table, then use the converter.
- You cannot use a code mapping table unless you first save the table using the CommuniNet code mapping utility after the installation of CommuniNet. Before using the converter, use the CommuniNet code mapping utility to save a code mapping table.
- The filename of a CommuniNet code mapping table must be `CMAPEX.TBL`. Before using the converter, store the code mapping table under a Windows directory.
- The processing by the character code converter does not reflect any changes made in a code mapping table by the CommuniNet code mapping utility during processing.
- You cannot save error logs and UAP trace information for the character code convert.
- Issue the function for starting character code conversion (`dc_clt_codeconv_open()` or `CBLDCUTL('CNVOPN  ')`) only once and then execute character code conversion (`dc_clt_codeconv_exec()` or `CBLDCUTL('CNVEXEC ')`). Do not issue the function for starting character code conversion more than once to prevent memory shortage. If you issue two or more functions, issue one function for terminating character code conversion (`dc_clt_codeconv_close()` or `CBLDCUTL('CNVCLS  ')` for each of the issued functions.

2.9 Multi-threading

2.9.1 Overview of a CUP suitable for multi-threading

All service calls are executed serially. If a service that requires a long period of processing is called, the subsequent service calls will be placed in a wait state.

If you need to call a time-consuming service, multi-threading is useful. Multi-threading allows you to allocate an exclusive thread for a time-consuming service call. While executing a time-consuming service call with the thread, you can execute other service calls with another thread.

2.9.2 Execution of functions not suited to multi-threading

In a multi-thread environment, if the `dc_rpc_call_s` function and the `dc_clt_cltin_s` function are issued in different threads, they might operate incorrectly.

The following figures show examples of executing functions not suited to a multi-thread environment.

Figure 2-35: Execution of functions not suited to a multi-thread environment (example 1)

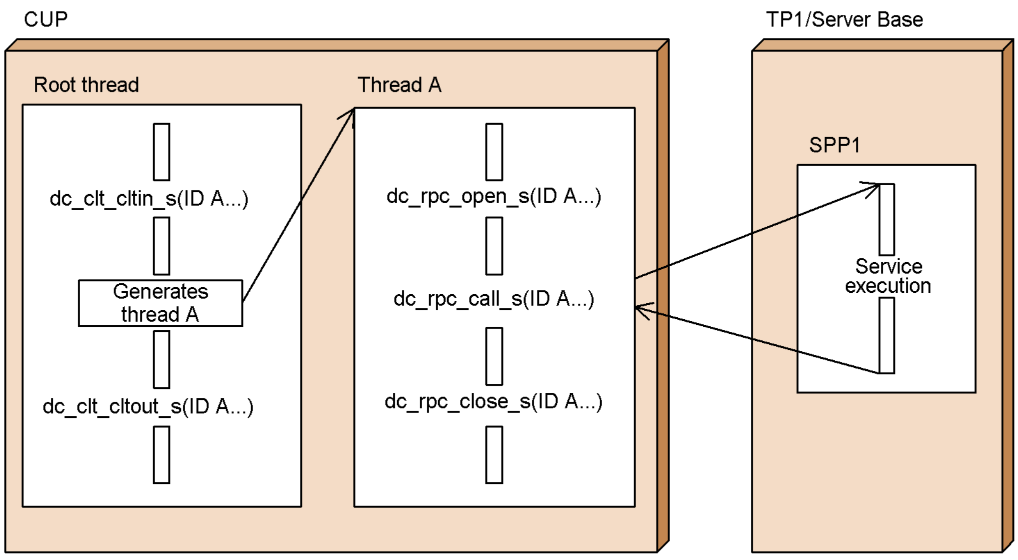
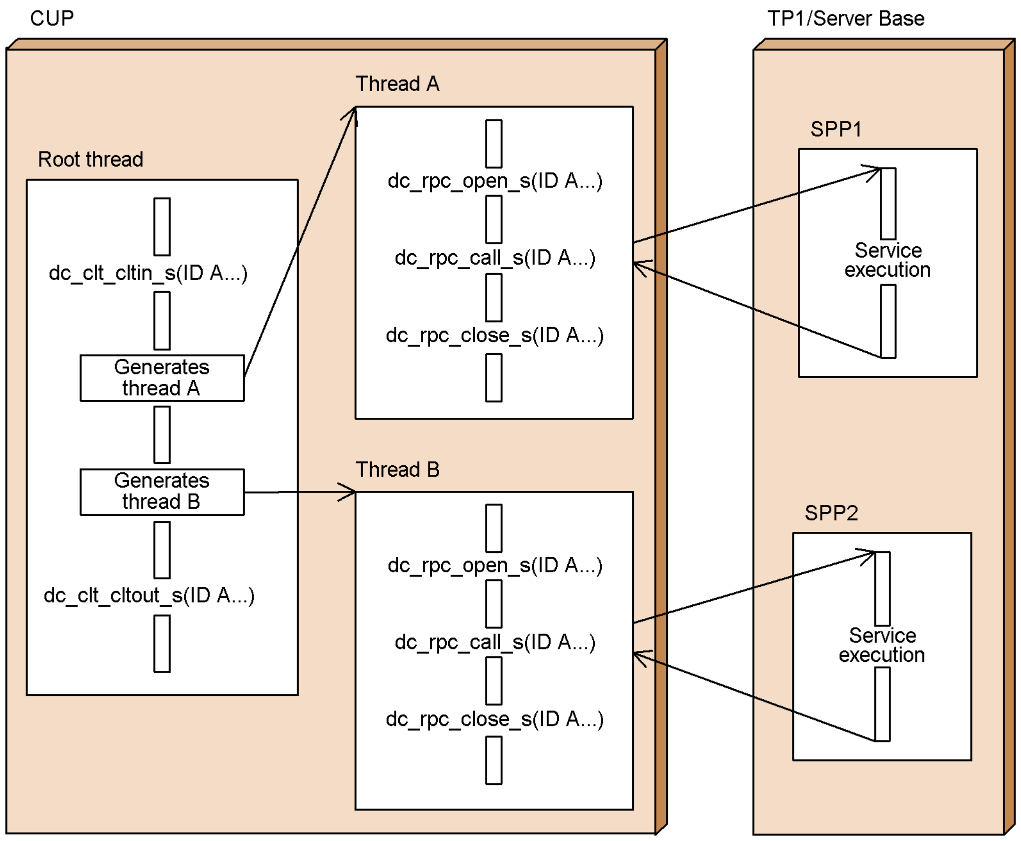


Figure 2-36: Execution of functions not suited to a multi-thread environment (example 2)



To ensure suitability when executing functions in a multi-thread environment, keep the following points in mind:

- In each thread, always issue the `dc_clt_cltin_s` function to receive a descriptor called a client ID. For all functions issued in all threads, specify the received client ID.

  A client ID that a process or thread has acquired through user authentication cannot be used in another process or thread.

- To execute RPCs concurrently in a multi-thread environment, an RPC must be executed from the `dc_clt_cltin_s` function for each thread.

  For the client ID argument of a function provided by TP1/Client, specify a client ID returned by the `dc_clt_cltin_s` function in the same thread. A client ID cannot be used across threads.

2.9.3 Notes on using multi-threading

- To execute a CUP in a multi-thread environment, use a function whose name ends with `_s`.
- While a function of TP1/Client is being executed (before the function returns) in a thread, do not execute another function of TP1/Client in the same thread.
- In each thread, always issue the `dc_clt_cltin_s` function to acquire a descriptor called a client ID, and specify the acquired client ID in all the functions issued in the thread.

  The client ID that a process or thread acquired by executing user authentication cannot be used in another process or thread.
- To execute multiple RPCs in a multi-thread environment concurrently, you must first issue the `dc_clt_cltin_s` function in each thread.

  In a thread, when you need to specify a client ID in an argument of a function provided by TP1/Client, specify the client ID returned by the `dc_clt_cltin_s` function issued in the thread. The client ID acquired in a thread is valid in the thread only.
- To create a client environment definition for each thread, specify a different file name in the `defpath` argument of the `dc_clt_cltin_s` function for each thread.

  If you want to use the facility for using a fixed reception port, a reception function of TCP/IP communication, or the facility for receiving one-way messages from the server, you must specify a different port number for each thread.
- If trace information for all threads is output to the same file, the CPU usage rate will increase, and the RPC throughput may degrade. Avoid this by using a different trace file for each process or thread.

2.10 Online tester

The user can use the online tester of TP1/Server Base. This function enables an SPP started from the CUP to be executed in test mode.

Before using the online tester, a test user ID must be specified in `DCUTOKEY` of the client environment definition. Also, the SPP started from the CUP must have a value other than 'no' specified in `test_mode` of the user service definition.

Using the online tester function requires TP1/Online Tester. For details of this function, see the manual *OpenTP1 Tester and UAP Trace User's Guide*.

When using the facility for establishing a permanent connection, you can use the online tester only within a transaction of a permanent connection established with the CUP executing process.

2.11 Troubleshooting

The troubleshooting functionality provides error logging, UAP trace collection, socket trace collection, and module trace collection. The information obtained is output to files. You must edit the information that is acquired using UAP trace collection, socket trace collection, and module trace collection, since it is output in binary format.

2.11.1 Error logging

Messages are output to an error log. There are situations in which the return value of a function is not enough to determine the cause of an error and resolve the problem. In such a case, examine the error log, as it may contain messages that will allow you to determine the cause of the error.

An error log is usually output to the CUP executing directory. However, when you specify `DCTRCPATH` in the client environment definition, the error log is output to the directory specified by `DCTRCPATH`. No error log is output when the specified directory does not exist.

Whether error logging is used depends on the specification of `DCTRCERR` in the client environment definition. If you omit the specification of `DCTRCERR`, or specify `DCTRCERR` to output an error log, two files (`dcerr1.trc` and `dcerr2.trc`) are created. When there is no information to be output, no file is created.

The two files are switched in accordance with the round-robin method, and are output chronologically. Thus, old information is deleted in units of files. If a write exceeds the file size specified for `DCTRCERR` in the client environment definition, the files are switched. If a write does not exceed the specified file size at the beginning of the write, the information is output. Thus, the actual file size may be greater than the specified file size.

Note that the character string output at the beginning of a file is maintenance information.

2.11.2 UAP trace collection

The information about the functions issued by users is output to a UAP trace. The UAP trace lets you check the sequence of the functions issued, the specified values of arguments, and the return values of functions.

A UAP trace is usually output to the CUP executing directory. However, when you specify `DCTRCPATH` in the client environment definition, the UAP trace is output to the directory specified by `DCTRCPATH`. No UAP trace is output when the specified directory does not exist.

Whether UAP trace collection is used depends on the specification of `DCTRCUAP` in the client environment definition. If you specify `DCTRCUAP` to output a UAP trace, two files (`dcuap1.trc` and `dcuap2.trc`) are created. If there is no information to be output, no file is created.

The two files are switched in accordance with the round-robin method, and are output chronologically. Thus, old information is deleted in units of files. If a write exceeds the file size specified for `DCTRCUAP` in the client environment definition, the files are switched. If a write does not exceed the specified file size at the beginning of the write, the information is output. Thus, the actual file size may be greater than the specified file size.

UAP traces are output in binary format. You must use the command for editing traces (`cltdump` command or `cltdmp32` command) to edit the UAP traces in text format.

2.11.3 Socket trace collection

TP1/Client outputs information about communication to a file as a socket trace. Although the contents of a socket trace are not disclosed, maintenance personnel may use them for troubleshooting.

Normally, the socket trace is output to the CUP execution directory. However, you can change the destination directory by specifying it in `DCTRCPATH` of the client environment definition. If the specified directory does not exist, the socket trace is not output.

Whether socket trace collection is used depends on the `DCTRCSOC` specification of the client environment definition. The file size also depends on the `DCTRCSOC` specification of the client environment definition. The data size is specified in `DCTRCSOCSIZE` of the client environment definition.

When socket trace collection is enabled, two files (`dcsoc1.trc` and `dcsoc2.trc`) are created. However, no files are created until there is information to be output.

Information is chronologically output to the two trace files on a round-robin basis. When the current destination file becomes full, the destination file is switched to the other. When the destination file is switched, the information existing in the new destination file will be deleted. TP1/Client checks the file size immediately before writing a new entry to the file. After the new entry is written, the file size may be larger than the predefined maximum, depending on the size of the new entry.

The socket trace files are binary files. To convert them into text files, use the `cltdump` or `cltdmp32` command.

The `DCTRCSOC` entry in the client environment definition determines whether the socket trace collection is enabled. The `DCTRCSOC` entry in the client environment definition specifies the file size. The `DCTRCSOCSIZE` entry in the client environment definition specifies the data size.

2.11.4 Module trace collection

TP1/Client outputs trace information about processing to a file as a module trace.

Although the contents of a module trace are not disclosed, maintenance personnel may use them for troubleshooting.

Normally, the module trace is output to the CUP executing directory. However, you can change the destination directory by specifying it in `DCTRCPATH` of the client environment definition. If the specified directory does not exist, the module trace is not output.

Whether module trace collection may be used depends on the `DCTRCMDL` of the client environment definition. When module trace collection is enabled, two files (`dcmdl1.trc` and `dcmdl2.trc`) are created. However, no files are created until there is information to be output.

Information is chronologically output to the two trace files on a round-robin basis. When the current destination file becomes full, the destination file is switched to the other. When the destination file is switched, the information existing in the new destination file will be deleted. TP1/Client checks the file size immediately before writing a new entry to the file. After the new entry is written, the file size may be larger than the predefined maximum, depending on the size of the new entry.

The module trace files are binary files. To convert them into text files, use the `cltdump` or `cltdmp32` command.

The `DCTRCMDL` entry in the client environment definition determines whether the module trace collection is enabled.

2.11.5 TP1/Server performance verification trace

A TP1/Server performance verification trace (PRF trace) provides trace information about major events related to services operating on TP1/Server. Use of this trace can improve the effectiveness of performance verification and troubleshooting.

In TP1/Client, identification information for performance verification can be preset. TP1/Client can add the identification information to the TP1/Server performance verification trace. The identification information for performance verification is also added to the TP1/Client UAP trace. This information can be used to check the TP1/Client function execution time (acquired by the UAP trace) against the TP1/Server service execution time (acquired by the performance verification trace). The information can also be used to determine the progress of processing.

(1) Transferring identification information for performance verification to TP1/Server

When `Y` is set for `DCCLTPRFINFOSEND` of the client environment definition, identification information for performance verification is transferred to TP1/Server.

However, TP1/Client might not be able to transfer the identification information depending on the TP1/Server version. For details, see the manual *OpenTP1 Operation* or the manual *TP1/LiNK User's Guide*.

(2) Checking the TP1/Server trace against the TP1/Client trace

When `Y` is specified for `DCCLTPRFINFOSEND` of the client environment definition, TP1/Client adds unique identification information (such as an IP address) to messages sent to TP1/Server every time the `dc_clt_cltin_s` function is issued. The added information is included in the TP1/Client UAP trace. The information is also included in the TP1/Server performance verification trace.

By checking the TP1/Client UAP trace against the TP1/Server performance verification trace, you can determine the sequence of processing between TP1/Client and TP1/Server.

Note, however, that if the `DCTRCUAP` specification of the client environment definition is disabled, the identification information for performance verification is not included in the UAP trace (the information is transferred to TP1/Server).

(3) Identification information for performance verification

Identification information for TP1/Client performance verification is added to the TP1/Client UAP trace and the TP1/Server performance verification trace.

(a) Items acquired as identification information for performance verification

The following explains the items acquired as identification information for performance verification:

Node ID: *aa*`[`*bb*`]` (four-byte alphanumeric character string)

aa: Either of the following values is set:

- For TP1/Client/W: `_W`
- For TP1/Client/P: `_P`

bb: Two alphanumeric characters (`0` to `9`, `A` to `Z`, `a` to `z`) are randomly set.

Root communication sequence number: `[`*xxxxxxxx*`]` (four-byte hexadecimal data)

xxxxxxxx: IP address

RPC communication sequence number: `[`*yyyy*`][`*zzzz*`]` (four-byte hexadecimal data)

yyyy: Random two-byte hexadecimal numbers

zzzz: Communication sequence number (incremented every time the function is called)

(b) Example of an output trace

The following shows an example of an output trace when the node ID is `_WOX`, the root communication sequence number is `f784d10a`, and the RPC sequence number is `4e880002`.

Example of the output of a TP1/Client UAP trace:

Information between the function entry (`EVENT=BEGIN`) and exit (`EVENT=END`) is acquired before a message is sent.

```
DATE = 2008/08/11 TIME = 05:08:35.603 PID = 2072:3152 SIZE =
26
FUNC = dc_rpc_call_s                         EVENT = PRF
Address   +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +a +b +c +d +e +f
                                         0123456789abcdef
00000000  5f 57 4f 58 2f 30 78 66 37 38 34 64 31 30 61 2f
                                          _WOX/0xf784d10a/
00000010  30 78 34 65 38 38 30 30 30 32
                                                0x4e880002
```

Example of the output of a TP1/Server performance verification trace:

```
PRF: Rec Node: smpl Run-ID: 0x4743dfcc Process: 53000
Trace: 4            Event: 0x1003 Time: 2008/08/11 05:08:36
583.000.000 Server-name: svgrp
Rc: 0 Client:  - 0x4e880002     Server: **** Root: _WOX -
0xf784d10a  Svc-Grp: ******************************** Svc:
****************************** Trn: *
```

(4) Trace information acquisition points

TP1/Client adds identification information for performance verification to the UAP trace when any of the following functions is issued:

- `dc_rpc_call_s`
- `dc_rpc_call_to_s`
- `dc_clt_connect_s`
- `dc_clt_disconnect_s`
- `dc_trn_begin_s`
- `dc_trn_chained_commit_s`
- `dc_trn_chained_rollback_s`
- `dc_trn_unchained_commit_s`
- `dc_trn_unchained_rollback_s`

2.11.6 Note on using the troubleshooting facility

When you use the `cltdump` or `cltdmp32` command for editing and outputting trace information, make sure that the version of the command and the version of TP1/Client are the same. If you use a command whose version differs from the version of TP1/Client, the execution results may be incorrect.

2.12 Host name extension

The maximum host name length that TP1/Client can handle is 63 characters. However, you can use the host name extension function to extend the host name length to a maximum of 255 characters. To use the function, specify `00000008` (logical addition) in `DCCLTOPTION` in the client environment definition.

The header file `dcvclt.h` provided by TP1/Client defines `MAXHOSTNAME` (=`64`) and `DCMAXDNSNAME` (=`256`). Use these definitions as required when creating a CUP.

2.12.1 Host name length and host name storage area length that can be specified in the arguments of C functions

The host name extension function extends the host name length and the host name storage area length that can be specified in the arguments of C functions.

The following table shows the host name lengths that can be specified in the arguments of C functions.

Table 2-9: Host name lengths that can be specified in the arguments of C functions

Function	Argument	Multiple hosts specifiable	Specifiable host name length	
			Not extended	Extended
`dc_clt_cltin_s`	`target_host`	Yes	63 characters (255 characters)#	255 characters (1023 characters)#
`dc_clt_set_raphost_s`	`raphost`			
`DCRPC_DIRECT_SCHEDULE`	`hostnm`	No	63 characters	255 characters
`dc_clt_cancel_notification_s`	`hostname`			
`dc_clt_send_s`	`hostname`			

Legend:

Yes: Multiple hosts can be specified.

No: Multiple hosts cannot be specified.

#

A number in parentheses indicates the maximum number of characters that can be specified in an argument (includes a port number and separators).

The following table shows the host name storage area length that can be specified in the arguments of C functions.

Table 2-10: Host name storage area lengths that can be specified in the arguments of C functions

Function	**Argument**	**Number of hosts to be stored**	**Host name storage area length**	
			Not extended (bytes)	**Extended (bytes)**
`dc_clt_get_raphost_s`	`raphost`	Multiple	256 or more	1024 or more
`dc_clt_cltin_s`	`set_host`	1	64 or more	256 or more
`dc_clt_accept_notification_s`	`hostname`			
`dc_clt_chained_accept_notification_s`	`hostname`			

Note

For the above arguments, you must prepare an area equal to or greater than the specification shown above. If the length of the area you prepare is smaller than the specification, the CUP may terminate abnormally.

2.12.2 COBOL-UAP creation programs when the host name extension function is used

If you want to use the host name extension function when creating a CUP in COBOL, use the following programs. For details about the data area length when the host name extension function is used, see the description of the data area where the UAP of each program sets values in *6. Request Statements Available for TP1/Client (COBOL Language)*.

Table 2-11: COBOL-UAP creation program that is called by the CALL statement used with the host name extension function

Function		COBOL-UAP creation program called by the CALL statement
User authentication	Requesting authentication of the client user	`CBLDCCLS('EXCLTIN ')`
		`CBLDCCLT('EXCLTIN ')`
Permanent connection	Specifying the destination of a request to establish a permanent connection	`CBLDCCLS('STRAPHST')`#
		`CBLDCCLT('STRAPHST')`#
	Acquiring the destination of a request to establish a permanent connection	`CBLDCCLS('GTRAPHST')`#
		`CBLDCCLT('GTRAPHST')`#
TCP/IP communication function	Sending messages	`CBLDCCLS('EXSEND  ')`
		`CBLDCCLT('EXSEND  ')`
Facility for receiving one-way messages from the server	Receiving one-way messages	`CBLDCCLS('EXNACPT ')`
		`CBLDCCLT('EXNACPT ')`
	Canceling the one-way message wait state	`CBLDCCLS('EXNCANCL')`
		`CBLDCCLT('EXNCANCL')`
	Receiving one-way messages	`CBLDCCLS('EXNCACPT')`
		`CBLDCCLT('EXNCACPT')`

#

When you specify `00000008` for `DCCLTOPTION` in the client environment definition, you must increase the size of the data area.

2.12.3 Number of characters that can be specified in an operand in the client environment definition

When using the host name extension function, you can specify a maximum of 255 characters for a host name specified in the following operands in the client environment definition:

- `DCHOST`#
- `DCCLTRAPHOST`#
- `DCCLTDCCMHOST`#

- `DCSNDHOST`
- `DCCLTCUPSNDHOST`

#

You can specify a maximum of 1023 characters in the operand (for other definitions, you can specify a maximum of 255 characters in the operand).

You can specify a host name with a maximum of 255 characters in a file specified for `DCCLTSERVICEGROUPLIST` in the client environment definition.

2.12.4 Notes on using the host name extension function

Note the following when specifying a host name:

- The host name cannot contain a colon (:) or a comma (,), which is used as a separator, or a semicolon (;), which is used for comments.
- The host name must consist of alphanumeric characters, of which at least one must be an alphabetic character.
- The host name must end with `\0`.

When you do not use the host name extension function, the host name can contain a maximum of 64 characters, including `\0`.

2.13 Send-host specification facility

When TP1/Client requests establishment of a connection to the server, the host that sends the request can be specified. This functionality is called the *send-host specification facility*.

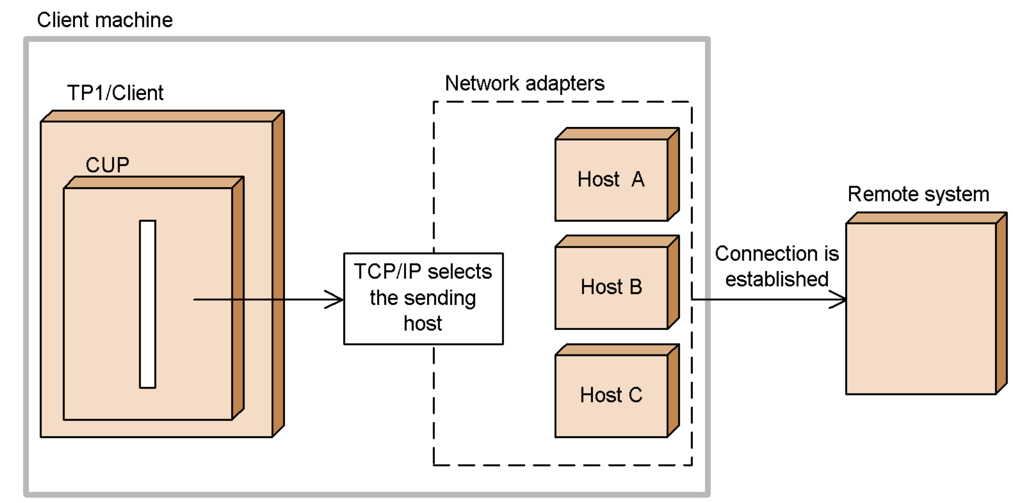
If multiple network adapters are connected to the host on which the CUP is operating, TCP/IP determines the host that the CUP uses to request establishment of its connections. However, if you use the send-host specification facility, you can specify the host that sends the connection establishment requests.

The sending host is specified for `DCCLTCUPSNDHOST` of the client environment definition.

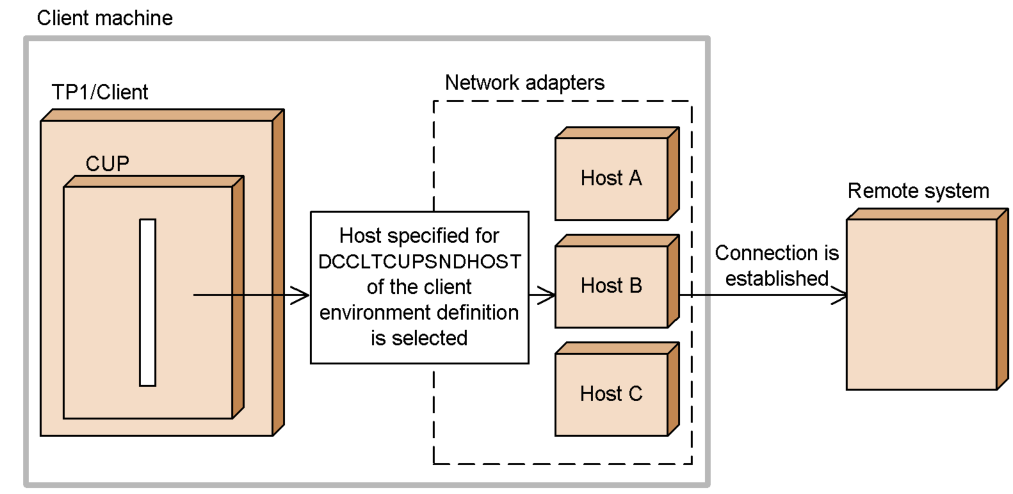
The following figure shows the difference when the send-host specification facility is used and when it is not used.

Figure 2-37: Difference when the send-host specification facility is used and when it is not used

● When the send-host specification facility for the CUP is not used



● When the send-host specification facility for the CUP is used


Client machine
TP1/Client
CUP
Network adapters
Host specified for DCCLTCUPSNDHOST of the client environment definition is selected
Host A
Host B
Host C
Connection is established
Remote system

2.14 Fixed receive-port facility

TP1/Client includes functionality that fixes the receive port used to receive a connection establishment request from the server. This functionality is called the *fixed receive-port facility*.

This facility is used when TP1/Server requests establishment of a connection to TP1/Client through a firewall whose filter settings are specified to allow transmission to only a single specified receive port.

Before this facility can be used, a receive port must be specified for `DCCLTCUPRCVPORT` of the client environment definition.

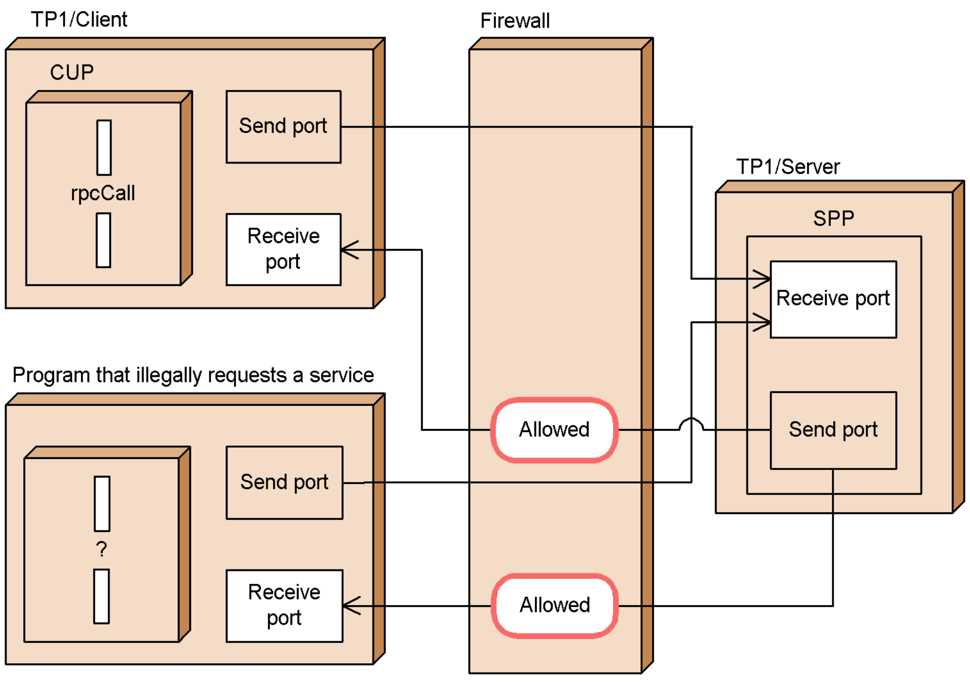
The following shows the difference when the fixed receive-port facility is not used and when it is used.

(1) When the fixed receive-port facility is not used

For communication of responses to RPCs, no filtering is performed on the receive port. The OS automatically assigns a port as the TP1/Client RPC receive port.

The following figure shows how RPCs that use the scheduler direct facility are processed when the fixed receive-port facility is not used.

Figure 2-38: Processing when the fixed receive-port facility is not used (for RPCs that use the scheduler direct facility)

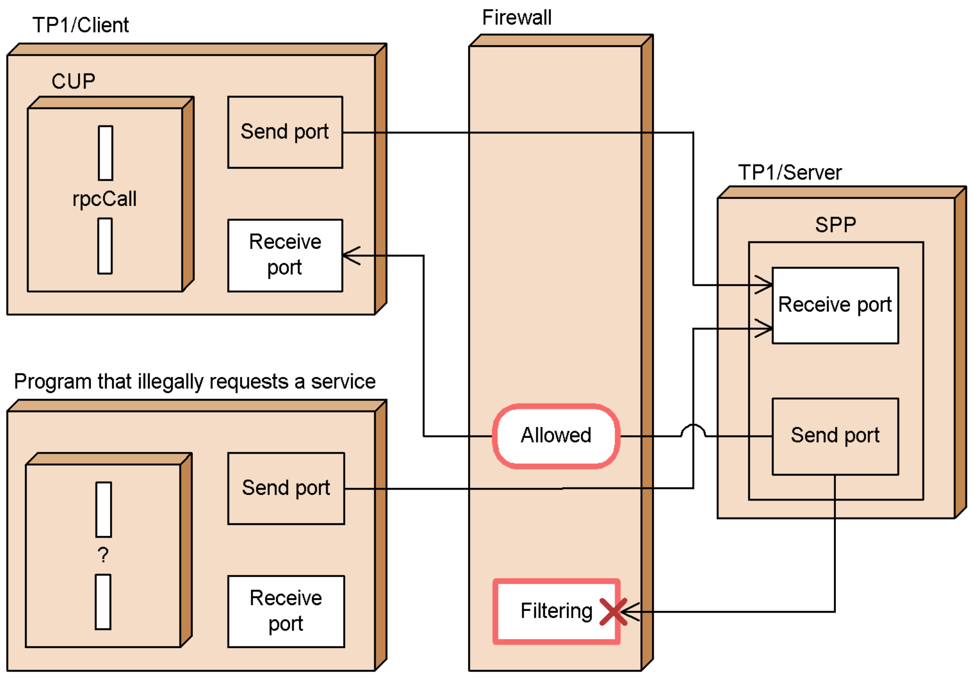


(2) When the fixed receive-port facility is used

The port specified for `DCCLCUPRCVPORT` of the client environment definition becomes the receive port on which the communication of responses to RPCs is allowed. All other ports are filtered out. Accordingly, the firewall is able to block responses from TP1/Server to illegal service requests.

The following figure shows the processing of RPCs that use the scheduler direct facility when the fixed receive-port facility is used.

Figure 2-39: Processing when the fixed receive-port facility is used (for RPCs that use the scheduler direct facility)



Note that the fixed receive-port facility is not enabled in the following cases:

- When the TCP/IP communication facility is used

  The receive port specified for `DCRCVPORT` of the client environment definition is used.

- When the facility for receiving one-way messages from the server is used

  The receive port specified in an argument of the `dc_clt_accept_notification_s` or `dc_clt_open_notification_s` function is used.

- When TP1/Client communicates with a RAP-processing server

  A connection established from TP1/Client is used for sending and receiving. Messages can safely pass through the firewall even when the fixed receive-port facility is not used.

Chapter

3. User Application Program Interface (C Language)

This chapter describes the function interface in the C language for user application programs. It also explains how to compile and link user application programs.

In this chapter, C functions (`dc_`*xxx*`_`*xxx*`_s`) are used to call DLLs. If you use functions of the normal object library (`dc_`*xxx*`_`*xxx*), replace the C function names with the corresponding function names of the normal object library.

This chapter contains the following sections:

3.1 Function interface

This section describes the functions used to interface with TP1/Client.

Develop CUPs using a C language compliant with ANSI C.

Like OpenTP1 service using programs (SUPs), CUPs do not use stubs. User programs must therefore convert data code (code structure and byte order).

SUP: Service Using Program

3.1.1 Table of functions

Table 3-1 lists TP1/Client functions.

You can use `dc_`*xxx*`_`*xxx*`_s` functions and character code conversion functions in a multi-thread environment. If both function versions (`dc_`*xxx*`_`*xxx*`_s` and `dc_`*xxx*`_`*xxx*) are supported, except for character code conversion functions, use the `dc_`*xxx*`_`*xxx*`_s` version of functions.

Since not all functions have the `dc_`*xxx*`_`*xxx*`_s` version, see the *Release Notes* that came with the program product to check whether the functions you want to use have the `dc_`*xxx*`_`*xxx*`_s` version.

For details about each function, see *4. TP1/Client Functions (C Language)*.

Table 3-1: Table of functions

TP1/Client Facility		Function Name
User authentication	Client user authentication requests	`dc_clt_cltin_s`
		`dc_clt_cltin`
	Release of client user authentication	`dc_clt_cltout_s`
		`dc_clt_cltout`
Remote procedure calls	UAP startup	`dc_rpc_open_s`
		`dc_rpc_open`
	UAP termination	`dc_rpc_close_s`
		`dc_rpc_close`
	Remote service requests	`dc_rpc_call_s`
		`dc_rpc_call`

TP1/Client Facility		Function Name
	Requesting a remote service with the communication destination specified	`dc_rpc_call_to_s`
		`dc_rpc_call_to`
	Updating the wait time for service response	`dc_rpc_set_watch_time_s`
		`dc_rpc_set_watch_time`
	Referencing the wait time for service response	`dc_rpc_get_watch_time_s`
		`dc_rpc_get_watch_time`
	Creating the `DCRPC_BINDING_TBL` structure	`DCRPC_DIRECT_SCHEDULE`
Permanent connection	Establishing permanent connection	`dc_clt_connect_s`
		`dc_clt_connect`
	Releasing permanent connection	`dc_clt_disconnect_s`
		`dc_clt_disconnect`
	Setting the destination of a request to establish a permanent connection	`dc_clt_set_raphost_s`
		`dc_clt_set_raphost`
	Acquiring the destination of a request to establish a permanent connection	`dc_clt_get_raphost_s`
		`dc_clt_get_raphost`
	Setting terminal identification information	`dc_clt_set_connect_inf_s`
		`dc_clt_set_connect_inf`
Transaction control	Transaction startup	`dc_trn_begin_s`
		`dc_trn_begin`
	Commit in chained mode	`dc_trn_chained_commit_s`

TP1/Client Facility		Function Name
		`dc_trn_chained_commit`
	Rollback in chained mode	`dc_trn_chained_rollback_s`
		`dc_trn_chained_rollback`
	Commit in unchained mode	`dc_trn_unchained_commit_s`
		`dc_trn_unchained_commit`
	Rollback in unchained mode	`dc_trn_unchained_rollback_s`
		`dc_trn_unchained_rollback`
	Post information about current transaction	`dc_trn_info_s`
		`dc_trn_info`
	Collection of identifiers for current transaction	`dc_clt_get_trnid_s`
		`dc_clt_get_trnid`
TCP/IP communication function	Sending messages	`dc_clt_send_s`
		`dc_clt_send`
	Receiving messages	`dc_clt_receive_s`
		`dc_clt_receive`
	Receiving messages (messages receivable even if an error occurs)	`dc_clt_receive2_s`
		`dc_clt_receive2`
	Sending assembled messages	`dc_clt_assem_send_s`
		`dc_clt_assem_send`
	Receiving assembled messages	`dc_clt_assem_receive_s`
		`dc_clt_assem_receive`

TP1/Client Facility		Function Name
Facility for receiving one-way messages from the server	Receiving one-way messages from the server to the client	`dc_clt_accept_notification_s`
		`dc_clt_accept_notification`
	Canceling wait for one-way messages	`dc_clt_cancel_notification_s`
		`dc_clt_cancel_notification`
	Starting reception of one-way messages	`dc_clt_open_notification_s`
		`dc_clt_open_notification`
	Terminating reception of one-way messages	`dc_clt_close_notification_s`
		`dc_clt_close_notification`
	Receiving a one-way message	`dc_clt_chained_accept_notification_s`
		`dc_clt_chained_accept_notification`
XATMI interface facility	Allocating typed buffer	`tpalloc`
	Releasing typed buffer	`tpfree`
	Establishing connection with interactive service	`tpconnect`
	Disconnecting connection with interactive service	`tpdiscon`
	Sending messages to interactive service	`tpsend`
	Receiving messages from interactive service	`tprecv`
Character code converter (When not using a code mapping table)#	Character code converter	`dc_clt_code_convert`
Character code converter (When using a code mapping table)#	Starting character code conversion	`dc_clt_codeconv_open`

TP1/Client Facility		Function Name
	Terminating character code conversion	`dc_clt_codeconv_close`
	Executing character code conversion	`dc_clt_codeconv_exec`

# This feature is only available for TP1/Client/P.

3.1.2 Format of function descriptions

TP1/Client functions are described in the following format:

Form

Shows the definition form of the TP1/Client library function and the argument data types. Use the listed data types when setting arguments for the function.

Purpose

Describes what the function does.

Arguments set by UAPs

Shows the arguments that must be set at function execution. Set each argument according to the given description.

Arguments that contain return values

Shows the arguments that reference the values returned by OpenTP1, server UAP, and TP1/Client when the function was previously executed.

Return values

Table of return values indicating whether the function executed correctly. If an error occurred, the return value indicates the error type.

When developing a UAP, always use the listed definition names rather than the numerical values. Return value definition names are defined in the header file.

Notes

Precautions on using the function.

(1) Symbols used for describing the values specified as arguments

The following tables lists the symbols that are used for describing the values specified as function arguments.

Symbol	Description
{ }	Select one of the items enclosed between braces. Example: `{DCCLT_CNV_EBCDIC|DCCLT_CNV_EBCDIK}` In this case, specify either `DCCLT_CNV_EBCDIC` or `DCCLT_CNV_EBCDIK`.
[ ]	The item enclosed between brackets can be omitted. Example: `[DCNOFLAGS]` `DCNOFLAGS` can be omitted.
_ (underscore)	When all the items enclosed between brackets are omitted, TP1/Client assumes the default indicated by an underscore. Example: `[{DCCLT_CNV_SPCHAN|DCCLT_CNV_SPCZEN}]` When both `DCCLT_CNV_SPCHAN` and `DCCLT_CNV_SPCZEN` are omitted, `DCCLT_CNV_SPCZEN` is assumed.
...	This symbol indicates a description is omitted. The item immediately before this symbol can be specified more than once consecutively. Example: *host-name* [:*port-number*][, *host-name* [:*port-number*],...] "*host-name* [:*port-number*]" can be specified more than once consecutively.
~	The item before this symbol conforms to the rule indicated between < > or (( )) described after ~.
<character string>	Any character(s)
<unsigned integer>	Numbers 0 to 9
(( ))	The specification range of the specified value is indicated.

(2) Description of the symbols specified as arguments

The following table lists the symbols specified as arguments.

Symbol	Description

| (Stroke)	This symbol delimits the items that are specified in a single argument. Insert the symbol between items.
	Example:
	`{DCCLT_CNV_EBCDIC|DCCLT_CNV_EBCDIK}`
	`[|{DCCLT_CNV_SPCHAN|DCCLT_CNV_SPCZEN}]`
	When specifying `DCCLT_CNV_EBCDIC` and `DCCLT_CNV_SPCHAN`, specify "`DCCLT_CNV_EBCDIC|DCCLT_CNV_SPCHAN`".

3.2 Compiling and linking user application programs

Compiling and linking methods differ according to the operating system environment.

3.2.1 Compiling and linking in UNIX environment

(1) Compiling

Write the CUP in an ANSI C compliant C language. Compile the CUP's C source program to create an object file. Use the `cc` command for TP1/Client/W.

Table 3-2 shows the compiler options that must be set.

Table 3-2: Required compiler options (in HI-UX/WE2, HP-UX, and non-Windows environments)

TP1/Client version	Option	Meaning
TP1/Client/W	`-Aa`	Compile as ANSI C.

Example

C language UAP source programs:

- `cupmain.c` (main function)
- `cupfnc1.c` (internal function 1)
- `cupfnc2.c` (internal function 2)

Compile each source program as shown below.

TP1/Client/W

```
cc -c -I/usr/include -Aa cupmain.c
cc -c -I/usr/include -Aa cupfnc1.c
cc -c -I/usr/include -Aa cupfnc2.c
```

When the source programs contain `dc_`*xxx_xxx*`_s` functions that support a multi-thread environment, enter commands as shown below.

```
xlc_r -c cupmain.c
xlc_r -c cupfunc1.c
xlc_r -c cupfunc2.c
```

Executing the above `cc` commands produces the following object files:

- `cupmain.o` (object file containing the main function)

- `cupfnc1.o` (object file containing internal function 1)
- `cupfnc2.o` (object file containing internal function 2)

(2) Linking

The CUP executable file is created by linking the files shown below. Use the `cc` command for TP1/Client/W.

- CUP object files (main function and internal functions)
- TP1/Client library

Example command lines for linking the above files are shown below.

Example

Creating the CUP executable file "example"

- Main function object file

  `cupmain.o`
- Internal function object files

  `cupfncl.o` and `cupfnc2.o`

Link the files as shown below.

TP1/Client/W

```
cc -o example cupmain.o cupfnc1.o cupfnc2.o
   -L/usr/lib -lclt
```

To create a CUP that supports a multi-thread environment, link the files by using the following command:

```
xlc_r -o example cupmain.o cupfnc1.o cupfnc2.o -L/usr/lib
-lclt
```

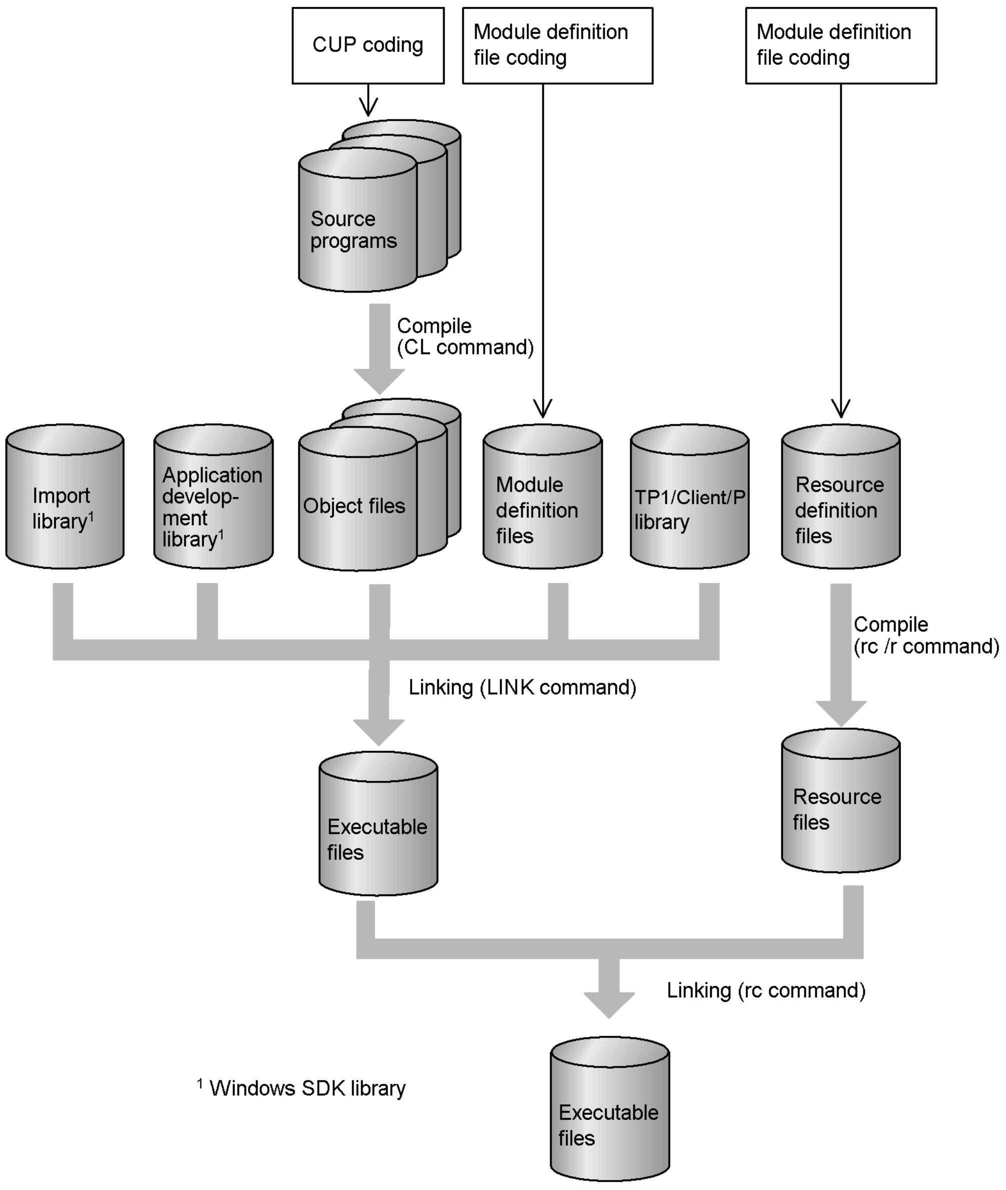
The `-L` option can be omitted.

3.2.2 Compiling and linking in Windows environments

(1) Procedure

The following figure shows the procedure for creating a CUP.

Figure 3-1: Procedure for creating a CUP



[1] Windows SDK library

(2) Compiling and linking

(a) Compiling the source program

Use the Microsoft C compiler (version 6.0 or later) to create CUP object files in the Windows environment. Use the `cc` command for compilation.

The compiler options required when normal object libraries are used differ from those required when DLLs are used. The following tables show the options required in these two cases.

Table 3-3: Required compiler options (Windows environment and normal object libraries)

TP1/Client version	Option	Meaning
TP1/Client/P	`/AM`	Specifies the memory model. `/AM`: Medium memory model (The TP1/Client/P library uses the medium memory model for Windows environments.)
	`/Zp`	Packs structures.
	`/Gw`	Creates Windows-specific prolog and epilog.
	`/DDCCLTFAR`	Defines the pointer as a far pointer.

Note

When the `XATMI` interface facility is used, `/DDCCLTDLL` must also be specified.

Table 3-4: Required compiler options (Windows environment and DLLs)

TP1/Client version	Option	Meaning
TP1/Client/P	`/Zp`	Packs the structure.
	`/Gw`	Generates the prolog and epilog dedicated to Windows.
	`/DDCCLTFAR`	Defines the pointer as a far pointer.
	`/DDCCLTDLL`	Develops a header file provided by TP1/Client/P for a DLL.

Example command lines for compiling a source program are shown below.

Example

C language UAP source programs:

- `cup.c` (main function)
- `cupsub1.c` (internal function 1)

- `cupsub2.c` (internal function 2)

Compile each source program as shown below.

Normal object library

```
CL /AM /Zp /Gw /DDCCLTFAR /c cup.c
CL /AM /Zp /Gw /DDCCLTFAR /c cupsub1.c
CL /AM /Zp /Gw /DDCCLTFAR /c cupsub2.c
```

When the XATMI interface facility is used, `/DDCCLTDLL` must also be specified.

DLL

```
CL /Zp /Gw /DDCCLTFAR /DDCCLTDLL /c cup.c
CL /Zp /Gw /DDCCLTFAR /DDCCLTDLL /c cupsub1.c
CL /Zp /Gw /DDCCLTFAR /DDCCLTDLL /c cupsub2.c
```

Executing the above `CL` commands produces the following object files:

- `cup.obj` (object file containing the main function)
- `cupsub1.obj` (object file containing internal function 1)
- `cupsub2.obj` (object file containing internal function 2)

(b) Creating the resource definition file

In this example, an icon is defined as a resource. Create the resource definition file `cup.rc` as follows:

```
CUPI   ICON cup.ico
```

`CUPI` is an arbitrary name given to the icon. Use the `SDKPAINT.EXE` tool included in Windows SDK to create the icon file (`cup.ico`).

(c) Compiling resources

Compile the resource definition file (`cup.rc`) as follows using the Windows SDK resource compiler:

```
rc /r cup.rc
```

Executing the above `rc` command creates the resource file `cup.res`.

(d) Creating a module definition file

Create the example module definition file `cup.def` as follows:

```
NAME        CUPEXEC
DESCRIPTION 'CUP SAMPLE PROGRAM'
EXETYPE     WINDOWS
STUB        'WINSTUB.EXE'
CODE PRELOAD MOVEABLE
DATA PRELOAD MOVEABLE
HEAPSIZE     1024
STACKSIZE    8192
```

Specify 8192 or more for `STACKSIZE`.

(e) Linking the CUP

Use the `LINK` command to link the files shown below and create the CUP executable file.

- CUP object files (main function and internal functions)
- TP1/Client/P library (`CLTW32.LIB` or `CLTWS32.LIB`)#
- Import library (provided by Windows SDK)
- Library for Windows application development (provided by Windows SDK)
- Module definition file

# `CLTCNV32.LIB` is also required to use the character code converter.

3.3 Example of user application program development

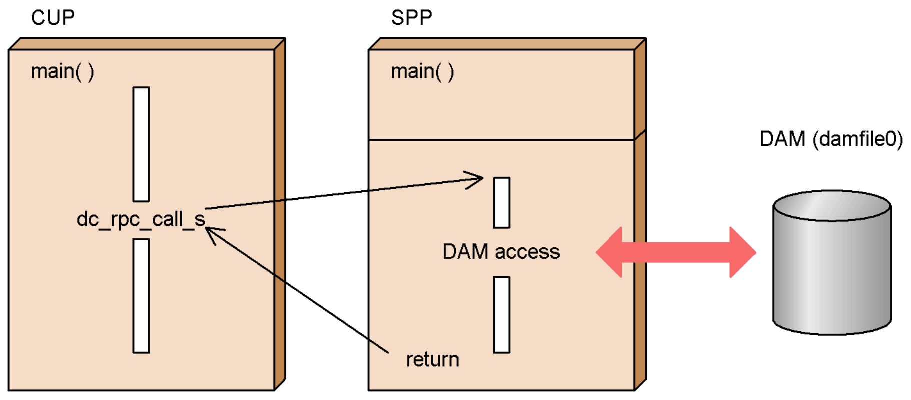
This section uses examples to describe how to code CUPs and SPPs when creating a UAP.

3.3.1 Creating CUPs and SPPs

The following figure shows the structure of the example CUP and SPP described in this section.

This example is for a non-Windows environment.

Figure 3-2: Example of CUP and SPP structure



The following shows how the example CUP is coded.

```
#include <stdio.h>
#include <string.h>
#include <dcvclt.h>
#include <dcvrpc.h>

#define BUFSIZE    512
#define SERVICE    "spp01"

main()
{
  char    in[BUFSIZE];
  DCULONG    in_len;
  char    out[BUFSIZE];
  DCULONG    out_len;
  char    indata[BUFSIZE];
  DCLONG      rc;
  DCCLT_ID  cltid;
  char    clt_flag = 0;
  char    rpc_flag = 0;

  /*
   *  Client user authentication request
   */
  if((rc = dc_clt_cltin_s(NULL, &cltid, NULL, NULL, "user01", "puser01",
      NULL, DCNOFLAGS)) != DC_OK){
    printf("cup01: dc_clt_cltin_s failed. CODE=%d\n", rc);
    goto PROG_EXIT;
  }
  clt_flag = 1;

  /*
   *  RPC-OPEN(RPC-environment initialization)
   */
  if((rc = dc_rpc_open_s(cltid, DCNOFLAGS)) != DC_OK) {
    printf("cup01: dc_rpc_open_s failed. CODE=%d\n", rc);
    goto PROG_END;
  }
  rpc_flag = 1;

  while (1) {
    printf("****** Messages Menu ******\n");
    printf("Retrieve message ... [1]    Write message ... [2]\n");
    printf("End ............. [9]\n");
    printf("Enter a number. =>");
    gets(indata);

    if(indata[0] == '1') {

      /*
       *  RPC-CALL(RPC execution)
       */
      strcpy(in, "cup01");
      in_len = strlen(in) + 1;
      out_len = sizeof(out);
      if((rc = dc_rpc_call_s(cltid, SERVICE, "get", in, &in_len, out,
          &out_len, DCNOFLAGS)) != DC_OK) {
        printf("cup01: dc_rpc_call_s failed. CODE=%d\n", rc);
        goto PROG_END;
      }
      printf("Message text : %s\n", out);
    }

    else if(indata[0] == '2') {
      printf("Enter the message. =>");
      gets(indata);
      if(indata[0] == '\0') {
        strcpy(indata, "No message was entered. \n");
      }

      /*
       *  RPC-CALL(RPC execution)
       */
      strcpy(in, indata);
      in_len = strlen(in) + 1;
      out_len = sizeof(out);
     if((rc = dc_rpc_call_s(cltid, SERVICE, "put", in, &in_len, out,
          &out_len, DCNOFLAGS)) != DC_OK) {
      printf("cup01: dc_rpc_call_s failed. CODE=%d\n", rc);
        goto PROG_END;
      }
      printf("%s\n", out);
    }

    else if(indata[0] == '9') {
      break;
    }

    else {
      continue;
    }
  }

PROG_END:
  /*
   *  RPC-CLOSE(RPC environment release)
   */
  if(rpc_flag) {
  dc_rpc_close_s(cltid, DCNOFLAGS);
  }

PROG_EXIT:
  if(clt_flag) {
  dc_clt_cltout_s(cltid, DCNOFLAGS);
  }
  exit(0);
}
```

The following shows how the example SPP (main function and DAM access) is coded.

```
#include <stdio.h>
#include <dcrpc.h>
#include <dcdam.h>
#define DAMFILE "damfile0"

int damfd;

main()
{
  int rc;

  /*
   *  RPC-OPEN (UAP startup)
   */
   if ((rc = dc_rpc_open(DCNOFLAGS)) != DC_OK) {
      printf("spp01:dc_rpc_open failed. CODE=%d\n", rc);
       goto PROG_END;
   }
   /*
    *  DAM-OPEN (Open logical file.)
    */
  if ((rc = dc_dam_open(DAMFILE,DCDAM_BLOCK_EXCLUSIVE)) < 0) {
      printf("spp01:dc_dam_open failed. CODE=%d\n", rc);
       goto PROG_END;
   }
   damfd = rc;
   /*
    *  RPC-MAINLOOP (SPP service startup)
    */
   printf("spp01:Entering mainloop. \n");
   if ((rc = dc_rpc_mainloop(DCNOFLAGS)) != DC_OK) {
      printf("spp01:dc_rpc_mainloop failed. CODE=%d\n", rc);
   }
   /*
    *  DAM-CLOSE (Close logical file.)
    */
   if ((rc = dc_dam_close(damfd,  DCNOFLAGS)) != DC_OK) {
     printf("spp01:dc_dam_close failed. CODE=%d\n", rc);
   }
PROG_END:
  /*
   *  RPC-CLOSE (UAP termination)
   */
  dc_rpc_close(DCNOFLAGS);
  printf("spp01:SPP service processing has finished. \n");
  exit(0);
}
```

The following shows how the example SPP (service function and DAM access) is coded.

```
#include <stdio.h>
#include <string.h>
#include <dcrpc.h>
#include <dctrn.h>
#include <dcdam.h>
#define DAMBLKSIZE  504

extern int  damfd;
static char damblk[DAMBLKSIZE];

void get(in, in_len, out, out_len)
    char              *in;
    unsigned long     *in_len;
    char              *out;
    unsigned long     *out_len;
{
    int               rc;
    struct DC_DAMKEY  keyptr;
    static char       *service = "get";

  printf("%s:Received a service request from %s. \n", service, in);

  /*
   *  TRN-BEGIN (Transaction startup)
   */
  if ((rc = dc_trn_begin()) != DC_OK) {
    sprintf(out,"%s:dc_trn_begin failed. CODE=%d\n",service,rc);
    printf("%s", out);
    goto PROG_END;
  }
  /*
   *  DAM_READ (DAM file read)
   */
  keyptr.fstblkno = 0;
  keyptr.endblkno = 0;
  if ((rc = dc_dam_read(damfd, &keyptr, 1, damblk,
           DAMBLKSIZE,  DCDAM_REFERENCE | DCDAM_NOWAIT)) != DC_OK) {
      sprintf(out, "%s:dc_dam_read failed. CODE=%d\n",service,rc);
      printf("%s", out);
      goto TRN_COMMIT;
  }
  strcpy(out, damblk);

TRN_COMMIT:
  /*
   * TRN_UNCHAINED_COMMIT (Unchained mode commit)
   */
  if ((rc = dc_trn_unchained_commit()) !=DC_OK) {
      sprintf(out, "%s:dc_trn_unchained_commit failed. CODE=%d\n",
              service, rc);
      printf("%s", out);
  }
PROG_END
    *out_len = strlen(out) + 1;
    return;
}

void put(in, in_len, out, out_len)
    char                  *in;
    unsigned long         *in_len;
    char                  *out;
    unsigned long         *out_len;
{
    int                   rc;
    struct DC_DAMKEY      keyptr;
    static char  *service = "put";

  printf("%s:Received a service request. \n", service);

  /*
   *  TRN-BEGIN (Transaction startup)
   */
  if ((rc = dc_trn_begin()) !=DC_OK) {
      sprintf(out, "%s:dc_trn_begin failed. CODE=%d\n",service,rc);
      printf("%s", out);
      goto PROG_END;
  }
  /*
   *  DAM_WRITE (DAM file write)
   */
  keyptr.fstblkno = 0;
  keyptr.endblkno = 0;
  strcpy(damblk, in);
  if ((rc = dc_dam_write(damfd, &keyptr, 1, damblk,
            DAMBLKSIZE, DCDAM_WAIT)) != DC_OK) {
    sprintf(out, "%s:dc_dam_write failed. CODE=%d\n",service,rc);
    printf("%s", out);
    dc_trn_unchained_rollback();
    goto PROG_END;
  }
  sprintf(out, "%s:Process completed normally. \n", service);
  /*
   *  TRN_UNCHAINED_COMMIT (Unchained mode commit)
   */
  if ((rc = dc_trn_unchained_commit()) != DC_OK) {
      sprintf(out, "%s:dc_trn_unchained_commit failed. CODE=%d\n",
              service, rc);
      printf("%s", out);
  }
PROG_END:
  *out_len = strlen(out) + 1;
  return;
}
```

3.3.2 Creating a user application program that supports a multi-thread environment

This subsection shows a coding example of a CUP that can operate in a multi-thread environment. This coding example is a program to call an SPP that echoes back a message sent from the CUP.

```
#include <stdio.h>
#include <dcvclt.h>
#include <dcvrpc.h>
#include <pthread.h>
#include <sys/errno.h>
#define BUFSIZE  512
#define SERVICE  "spp01"
#define THDMAX   5

void *CUP_thread(void *arg)
{
   char        in[BUFSIZE];
```

```
    DCULONG     in_len;
    char        out[BUFSIZE];
    DCULONG     out_len;
    int         rc = DC_OK;
    DCCLT_ID    cltid;
    int         myid;

    myid = (int)arg;

    /* Client user authentication request */
    if ((rc = dc_clt_cltin_s(NULL, &cltid, NULL, NULL,
                   "user01", "puser01", NULL, DCNOFLAGS)) != DC_OK) {
       printf("cup%d: dc_clt_cltin failed. CODE=%d\n", myid, rc);
       goto PROG_EXIT;
     }

     /* RPC-OPEN (RPC environment initialization) */
   if ((rc = dc_rpc_open_s(cltid, DCNOFLAGS)) != DC_OK) {
        printf("cup%d: dc_rpc_open failed. CODE=%d\n", myid, rc);
        goto PROG_END;
    }

    /* RPC-CALL (RPC execution) */---*/
    strcpy(in, "HELLO SPP !!");
    in_len = strlen(in) + 1;
    out_len = sizeof(out);
    if ((rc = dc_rpc_call_s(cltid, SERVICE, "echo", in, &in_len,
                            out, &out_len, DCNOFLAGS)) != DC_OK) {
        printf("cup%d: dc_rpc_call failed. CODE=%d\n", myid, rc);
        goto PROG_END;
    }
    printf("%s\n", out);
PROG_END:

    /* RPC-CLOSE (RPC environment release) */
    dc_rpc_close_s(cltid, DCNOFLAGS);

PROG_EXIT:
    /* Client user authentication release */
    dc_clt_cltout_s(cltid, DCNOFLAGS);

    /* Thread termination */
   pthread_exit(arg);
}

main()
{
   int        i;
   int        rc;
   int        exit_value;
   pthread_t  threads[THDMAX];

   /* Thread creation */
   for (i = 1; i < THDMAX; i++) {
      rc = pthread_create((pthread_t *)&threads[i],
                           NULL,
                           CUP_thread,
                           (void *)i);
      if (rc < 0) {
       printf("cup0: pthread_create failed. CODE=%d\n", errno);
       }
   }

   /* Wait for thread termination */
   for (i = 1; i < THDMAX; i++) {
     rc = pthread_join(threads[i], (void **)&exit_value);
      if (rc < 0) {
        printf("cup0: pthread_join failed.CODE=%d\n", errno);
       }
   }
}
```

Chapter

4. TP1/Client Functions (C Language)

This subsection describes the functions that can be used with TP1/Client.

The description in this chapter uses function names in `dc_`*xxx*`_`*xxx*`_s` format (`_s` version), the format used to call DLL functions in C. If you use the ordinary object library functions, replace the function names with the corresponding function names in `dc_`*xxx*`_`*xxx* format (non-`_s` version).

4.1 Notes on using functions

The following are notes on using functions:

- We recommend that you use the `_s` version of functions (`dc_`*xxx_xxx*`_s`), which can operate in a multi-thread environment. Note, however, that some TP1/Client program products might not support the `_s` version of functions. For the support status of the `_s` version of functions (`dc_`*xxx_xxx*`_s`), see the *Release Notes*.
- In TP1/Client/W, you do not need to specify the `CLTFAR` pointer (even though specifying the pointer does not result in any problem). However, in TP1/Client/P, you must specify the `CLTFAR` pointer. How the pointer operates is defined in `dcvclt.h`.
- For the non-`_s` version of functions (`dc_`*xxx_xxx*) in TP1/Client/P, you can specify either the `CLTFAR` pointer or the `far` pointer without any problems.
- The `_s` version of a function (`dc_`*xxx_xxx*`_s`) and the non-`_s` version of the corresponding function (`dc_`*xxx_xxx*) do not have the same number of arguments. In a multi-thread environment, you must use the `_s` version. The non-`_s` version of a function (`dc_`*xxx_xxx*) will not correctly operate in a multi-thread environment. Although the character code converter provides the non-`_s` version of functions only, the functions will operate correctly in a multi-thread environment.

4.2 User authentication

4.2.1 dc_clt_cltin_s - client user authentication request

(1) Form

(a) TP1/Client/W

■ **_s version of the function**

```
#include <dcvclt.h>
DCLONG dc_clt_cltin_s(HWND hWnd, DCCLT_ID *cltid,
                      char *defpath,
                      char *target_host,
                      char *logname,
                      char *passwd,
                      char *set_host, DCLONG flags)
```

■ **Non-_s version of the function**

```
#include <dcvclt.h>
int dc_clt_cltin(char *target_host,
                 char *logname,
                 char *passwd,
                 char *set_host, DCLONG flags)
```

(b) TP1/Client/P

■ **_s version of the function**

```
#include <dcvclt.h>
DCLONG dc_clt_cltin_s(HWND hWnd, DCCLT_ID CLTFAR *cltid,
                      char CLTFAR *defpath,
                      char CLTFAR *target_host,
                      char CLTFAR *logname,
                      char CLTFAR *passwd,
                      char CLTFAR *set_host, DCLONG flags)
```

■ **Non-_s version of the function**

```
#include <dcvclt.h>
int dc_clt_cltin(char CLTFAR *target_host,
                 char CLTFAR *logname,
                 char CLTFAR *passwd,
                 char CLTFAR *set_host, DCLONG flags)
```

(2) Purpose

Requests TP1/Server as a gateway to verify the client user specified by the login name.

Always execute the `dc_clt_cltin_s` function even when you suppress user authentication.

(3) Arguments set by UAPs

- `hWnd`

  Specify NULL.

- `cltid`

  Pointer to the area that receives the client ID (defined in the header file `dcvclt.h`; its type is `DCCLT_ID`).

  When authentication of the client user terminates normally, the client ID is set in the specified area. The client ID must not be destroyed before the `dc_clt_cltout_s` function is issued.

  The acquired client ID can only be used within the thread in which this function was issued. If you use `dc_rpc_call_s` or other functions to pass the client ID to another thread, the other thread may operate incorrectly.

- `defpath`

  Specify the path name of the client environment definition file. The path name must be specified with the full path or with a relative path from the current drive and the current directory. The following shows the order in which files are loaded when the path name is specified.

  - In TP1/Client/P

    Client environment definition files are loaded in the following order:

    1. The `BETRAN.INI` file in the Windows directory

    2. The client environment definition file specified in the `defpath` argument

    The definitions in both the client environment definition file and the `BETRAN.INI` file take effect.

    If the same definition is specified in each file with a different value, the value specified in the client environment definition file takes effect.

    If neither the client environment definition file nor the `BETRAN.INI` file contains the necessary specification, TP1/Client/P uses the defaults.

  - In TP1/Client/W

    All definitions specified in the environment variables will be invalid. TP1/Client/W uses the defaults for definitions that are not specified in the client environment definition file specified in the `defpath` argument.

  You can omit the path name by specifying NULL at the beginning of the `defpath` argument. The following describes the operation when the path name is omitted.

  - In TP1/Client/P

TP1/Client/P uses the `BETRAN.INI` file in the Windows directory as the client environment definition file. If the `BETRAN.INI` file does not exist or if the contents of the definition file are invalid, TP1/Client/P uses the defaults.

- In TP1/Client/W

  TP1/Client/W uses the values specified in the environment variables. If an environment variable is not specified, TP1/Client/W uses the default.

The following describes operation when the client environment definition file specified in the `defpath` argument does not exist or when the contents of the definition file are invalid.

- In TP1/Client/P

  TP1/Client/P uses the `BETRAN.INI` file in the Windows directory as the client environment definition file. If the `BETRAN.INI` file does not exist or if the contents of the definition file are invalid, TP1/Client/P uses the defaults.

- In TP1/Client/W

  TP1/Client/W uses the defaults. The values specified in the environment variables will be invalid.

■ `target_host`

Specify the host name and port number of TP1/Server that is used as a gateway when authentication is requested. Multiple TP1/Servers can be specified as gateways (use commas (,) to delimit them).

You can specify a maximum of 63# characters for the host name. When specifying multiple host names, you can specify a maximum of 255# characters, including port numbers, in the `target_host` argument.

Form:

host-computer-name`[:`*port-number*`][,`*host-computer-name*`[:`*port-numbe r*`],...]`

host-computer-name~<character string>

port-number~<unsigned integer>((5001-65535))

Do not place a null character (space or tab) except after the separator (,).

You can specify an IP address in decimal dot notation for the host name.

When the port number is omitted, the value for client environment definition `DCNAMPORT` is assumed.

When you have specified more than one TP1/Server in the `target_host` argument and an error is detected in the TP1/Server being used as a gateway, system operation depends on the specification of DCHOSTSELECT in the client environment definition. If N is specified for DCHOSTSELECT, the system attempts to replace the failed node by referencing the next TP1/Server of the currently used TP1/Server. If Y is specified for DCHOSTSELECT, the system selects a TP1/Server at random (excluding the TP1/Server in which the error was detected) and attempts to replace the failed node.

When NULL is specified, the function references client environment definition DCHOST. If target_host is NULL and DCHOST is not set, a broadcast is performed to determine the target host computer.

To perform a broadcast in TP1/Client/P, you must specify the broadcast address in the hosts file (the host name must be broadcast). If the host name is not specified, the dc_clt_cltin_s function returns a DCCLTER_SYSERR error.

# If you specify 00000008 for DCCLTOPTION in the client environment definition, you can specify a maximum of 255 characters for the host name. When specifying multiple host names, you can specify a maximum of 1023 characters, including port numbers, in the target_host argument.

- logname

  Specify the login name of the client user. The login name can have a maximum of 15 characters. To communicate with a server other than TP1/Server, specify a value other than NULL. If you specify NULL, the function returns an error.

- passwd

  Specify the password for the login name specified for logname. The password can have a maximum of 15 characters. Set passwd to NULL if not setting the password.

- set_host

  Pointer to a 64-byte# area containing the name of the host that actually received the client user authentication request. The host name is not stored if NULL is specified.

  # If you specify 00000008 for DCCLTOPTION in the client environment definition, this value is 256 bytes, not 64 bytes.

- flags

  Specify DCCLT_NO_AUTHENT to suppress user authentication for using the remote API facility. Specify DCNOFLAGS not to suppress user authentication.

(4) Arguments specifying the containers of returned values

- cltid

Specifies the area for containing the returned client ID.

- `set_host`

  Specifies the area for storing the returned host name (or IP address in decimal-dot notation) of the server that actually performed user authentication. Nothing is returned if you suppress user authentication.

(5) Return values

Return Value	Value (decimal)	Meaning
`DC_OK`	0	Normal termination
`DCCLTER_INVALID_ARGS`	-2501	Invalid argument
`DCCLTER_PROTO`	-2502	The `dc_clt_cltin` function has already been issued. This value is not returned if the `dc_clt_cltin_s` function is executed.
`DCCLTER_FATAL`	-2503	Channel initialization failed. Or, the client environment is wrongly specified.
`DCCLTER_NO_BUFS`	-2504	A necessary amount of buffer could not be allocated. Alternatively, the resource became insufficient.
`DCCLTER_NET_DOWN`	-2506	Communication fault
`DCCLTER_OLTF_NOT_UP`	-2515	OpenTP1 is not running on the requested service node.
`DCCLTER_SYSERR`	-2518	System error
`DCCLTER_REJECT`	-2527	The specified login name is not registered in the target host, or the password does not match. Alternatively, the OpenTP1 server may not support user authentication. Check whether `client_uid_check` is specified correctly in the system common definition.
`DCCLTER_PORT_IN_USE`	-2547	A specified port number is in use. Alternatively, port numbers that can be assigned automatically by the operating system are insufficient.

(6) Notes

- When TP1/Server runs on UNIX and an asterisk (`*`) is set in the encryption password field by the security facility, TP1/Server cannot perform user authentication. In this case, the `dc_clt_cltin_s` function returns a `DCCLTER_REJECT` error.
- If the `dc_clt_cltin_s` function returns an error, you cannot specify the `cltid` argument in a TP1/Client function to be issued because the specified `cltid` argument will be invalid. You will need to start over from execution of the

dc_clt_cltin_s function.

- When you specify the area for receiving the value of the set_host argument, you must specify at least 64 bytes#. If the area is smaller than 64 bytes#, the area may be corrupted during TP1/Client internal processing. The argument value is not stored when you suppress user authentication.

  # If you specify 00000008 for DCCLTOPTION in the client environment definition, this value is 256 bytes, not 64 bytes.

- In TP1/Client, you can use a different client environment definition for each dc_clt_cltin_s function call. To do so, create a separate client environment definition file for each dc_clt_cltin_s function call, and specify the file name in the defpath argument of the function.

4.2.2 dc_clt_cltout_s - release of client user authentication

(1) Form

TP1/Client/W or TP1/Client/P

(a) _s version of the function

```
#include <dcvclt.h>
void dc_clt_cltout_s(DCCLT_ID cltid, DCLONG flags)
```

(b) Non-_s version of the function

```
#include <dcvclt.h>
void dc_clt_cltout(DCLONG flags)
```

(2) Purpose

Releases a client user authentication. The CUP is no longer able to receive OpenTP1 services.

The dc_clt_cltout_s function must be issued before termination of a CUP. When issue, dc_clt_cltout_s function must be paired with the dc_clt_cltin_s function.

(3) Argument set by UAPs

- ■ cltid

  Set the client ID received by the dc_clt_cltin_s function.

- ■ flags

  Set DCNOFLAGS.

4.3 Remote procedure calls

4.3.1 dc_rpc_open_s - UAP startup

(1) Form

TP1/Client/W or TP1/Client/P

(a) _s version of the function

```
#include <dcvrpc.h>
DCLONG dc_rpc_open_s(DCCLT_ID cltid, DCLONG flags)
```

(b) Non-_s version of the function

```
#include <dcvrpc.h>
int dc_rpc_open(DCLONG flags)
```

(2) Purpose

Initializes the environment for calling OpenTP1 SPPs or using the TCP/IP communication function.

Execute the `dc_rpc_open_s` function before executing a remote procedure call, transaction control, or various functions for the transaction control.

(3) Argument set by UAPs

- `cltid`

  Set the client ID received by the `dc_clt_cltin_s` function.

- `flags`

  Specifies the environment to be initialized.

  `DCNOFLAGS`

  Environment for calling SPPs

  `DCCLT_ONEWAY_SND`

  Environment for sending send-only messages

  `DCCLT_ONEWAY_RCV`

  Environment for receiving receive-only messages

  `DCCLT_SNDRCV`

Environment for sending and receiving messages.

If DCNOFLAGS is specified, the TCP/IP communication function cannot be used.

Even if any other than DCNOFLAGS is specified, the RPC function can be used.

If DCCLT_SNDRCV is specified, the DCCLT_ONEWAY_SND and DCCLT_ONEWAY_RCV must not be specified at the same time.

(4) Return values

Return Value	Value (decimal)	Meaning
DC_OK	0	Normal termination
DCRPCER_INVALID_ARGS	-2401	Invalid argument
DCRPCER_PROTO	-2402	The dc_rpc_open_s function has already been issued. Alternatively, the dc_clt_cltin_s function has not been executed.
DCRPC_FATAL	-2403	The return value has been returned because of failure to initialize, or an invalid client environment definition.
DCRPCER_PORT_IN_USE	-2447	A specified port number is in use.
DCCLTER_INVALID_CLTID	-2544	The client ID specified in cltid differs from the client ID received by the dc_clt_cltin_s function.

(5) Notes

Issuing the dc_rpc_close_s function must not be immediately followed by issuing the dc_rpc_open_s function with flags set to DCCLT_ONEWAY_RCV, as the following describes. The dc_rpc_open_s function can only be issued 15-20 seconds later.

- After the dc_rpc_open_s function is issued with flags set to DCCLT_ONEWAY_RCV, the dc_clt_receive_s function has been issued and a message is being received. The CUP issues the dc_rpc_close_s function to free the connection before the remote system frees it.

4.3.2 dc_rpc_close_s - UAP termination

(1) Form

TP1/Client/W or TP1/Client/P

(a) _s version of the function

```
#include <dcvrpc.h>
void dc_rpc_close_s(DCCLT_ID cltid, DCLONG flags)
```

(b) Non-_s version of the function

```
#include <dcvrpc.h>
void dc_rpc_close(DCLONG flags)
```

(2) Purpose

Releases the environment for calling OpenTP1 SPPs or using the TCP/IP communication function.

When issued, the `dc_rpc_close_s` function must be paired with the `dc_rpc_open_s` function.

The functions that can be issued after the `dc_rpc_close_s` function are:

- `dc_rpc_open_s`
- `dc_clt_cltout_s`

(3) Argument set by UAPs

- `cltid`

  Set the client ID received by the `dc_clt_cltin_s` function.

- `flags`

  Set `DCNOFLAGS`.

(4) Note

The `dc_rpc_close_s` function does not return a value. Note that if an invalid value is specified in an argument, the environment will not be released.

4.3.3 dc_rpc_call_s - remote service request

(1) Form

(a) TP1/Client/W

■ _s version of the function

```
#include <dcvrpc.h>
DCLONG dc_rpc_call_s(DCCLT_ID cltid, char *group,
                     char *service, char *in,
                     DCULONG *in_len, char *out,
                     DCULONG *out_len, DCLONG flags)
```

■ Non-_s version of the function

```
#include <dcvrpc.h>
```

```
int dc_rpc_call(char *group, char *service,
                char *in, DCULONG *in_len,
                char *out, DCULONG *out_len, DCLONG flags)
```

(b) TP1/Client/P

■ _s version of the function

```
#include <dcvrpc.h>
DCLONG dc_rpc_call_s(DCCLT_ID cltid, char CLTFAR *group,
                  char CLTFAR *service, char CLTFAR *in,
                  DCULONG CLTFAR *in_len, char CLTFAR *out,
                  DCULONG CLTFAR *out_len, DCLONG flags)
```

■ Non-_s version of the function

```
#include <dcvrpc.h>
int dc_rpc_call(char CLTFAR *group, char CLTFAR *service,
                char CLTFAR *in, DCULONG CLTFAR *in_len,
                char CLTFAR *out, DCULONG CLTFAR *out_len,
                DCLONG flags)
```

(2) Purpose

Requests an SPP service. Calls a service function, specifying the service group name and service name, and receives a response from the service function.

OpenTP1 must be running on the server UAP node to which the service request is sent. The `dc_rpc_call_s` function returns a `DCRPCER_NET_DOWN`, `DCRPCER_OLTF_NOT_UP`, or `DCRPCER_OLTF_INITIALIZING` error if OpenTP1 is not running (or is initializing).

The function returns a `DCRPCER_SERVICE_CLOSED` error if the target service group is shut down when the `dc_rpc_call_s` function is executed.

If the target service group is terminating or has been terminated by commands such as the `dcsvstop` command, the function returns a `DCRPCER_SERVICE_TERMINATING`, `DCRPCER_SERVICE_CLOSED`, or `DCRPCER_NO_SUCH_SERVICE_GROUP` error. The actual value returned depends on the timing of the `dc_rpc_call_s` function call.

A socket-receiving type server concurrently controls messages by specifying `max_socket_msg` and `max_socket_msglen` in the user service definition. This sometimes prevents the user from receiving a service request. If a service request cannot be received, the `dc_rpc_call_s` function returns with a `DCRPCER_SERVER_BUSY` error. If this value is returned, the CUP may make a service request by reexecuting the process after a certain time.

To communicate with XDM/DCCM3 in the normal communication mode, specify the host name and port number of the XDM/DCCM3 logical terminal in `DCCLTSERVICEGROUPLIST` in the client environment definition, and then execute the `dc_rpc_call_s` function.

(a) Setting arguments

The CUP reserves an area (`out`) for the service function response. The CUP also sets the following values for the `dc_rpc_call_s` function:

- Input parameter (`in`)
- Input parameter length (`in_len`)
- Response length (`out_len`)

The input parameter, input parameter length, and response length values set for the `dc_rpc_call_s` function by the CUP are transferred to the service function without modification. The response length is ignored if the function requests a no-response type service.

The maximum values for `in_len` and `out_len` are defined with `DCRPC_MAX_MESSAGE_SIZE`# in the `dcvrpc.h` header file.

# If you specify 2 or a larger value for `DCCLTRPCMAXMSGSIZE` in the client environment definition, the value you specify is used rather than the value of `DCRPC_MAX_MESSAGE_SIZE` (1 megabyte).

(b) Referencing arguments

The following values are available after the completion of service function processing.

- Service function response (`out`)
- Service function response length (`out_len`)

`out_len` contains the actual length of the response returned from the service function.

For synchronous response type RPCs (`DCNOFLAGS` set in flags), out and `out_len` can be referenced after the `dc_rpc_call_s` function returns. For no-response type RPCs (`DCRPC_NOREPLY` set in flags), out and `out_len` cannot be referenced. Also, out and out_len cannot be referenced when the `dc_rpc_call` or function returns an error.

The function returns a `DCRPCER_REPLY_TOO_BIG` error if the response is larger than the response area (`out`) reserved by the CUP.

(3) Arguments set by UAPs

■ `cltid`

Set the client ID received by the `dc_clt_cltin_s` function.

■ `group`

Set the service group name as a null-terminated character string up to 31 bytes in length.

- `service`

  Set the service name as a null-terminated character string up to 31 bytes in length.

- `in`

  Set the service input parameter.

- `in_len`

  Set the length of the service input parameter as a value between 1 and `DCRPC_MAX_MESSAGE_SIZE`#.

  # If you specify 2 or a larger value for `DCCLTRPCMAXMSGSIZE` in the client environment definition, the value you specify is used rather than the value of `DCRPC_MAX_MESSAGE_SIZE` (1 megabyte).

- `out`

  Set the area for receiving the service response.

- `out_len`

  Set the length of the service response as a value between 1 and `DCRPC_MAX_MESSAGE_SIZE`#.

  # If you specify 2 or a larger value for `DCCLTRPCMAXMSGSIZE` in the client environment definition, the value you specify is used rather than the value of `DCRPC_MAX_MESSAGE_SIZE` (1 megabyte).

- `flags`

  Set the RPC type.

  `DCNOFLAGS`

  Synchronous response type RPC

  `DCRPC_NOREPLY`

  No-response type RPC

  `DCRPC_CHAINED`

  Chained RPC

  If `DCNOFLAGS` or `DCRPC_CHAINED` is set, the `dc_rpc_call_s` function does not return until a response is received or until a response timeout error occurs. The response timeout period is specified by `DCWATCHTIM` in the client environment definition. When the request destination SPP aborts, the function immediately returns an error. The error accompanies either of the following values depending on the response wait time specified for DCWATCHTIM.

- DCWATCHTIM = 1-65535: DCRPCER_TIMED_OUT
- DCWATCHTIM = 0 (wait indefinitely): DCRPCER_SERVICE_NOT_UP

You can change the response wait time while the CUP is executing. To do this, execute the dc_rpc_set_watch_time_s function before executing the dc_rpc_call_s function.

You can specify DCRPC_CHAINED only when a transaction or permanent connection is active.

If DCRPC_NOREPLY is set, the requested service is treated as a no-response type service. The dc_rpc_call_s function returns immediately without waiting for the service to complete execution. The response (out) and response length (out_len) cannot be referenced. Also, the CUP cannot determine whether the service function actually executed or not.

An RPC from transaction processing can be changed to a service request for no-transaction. If DCRPC_TPNOTRAN is specified in the RPC type parameter, the service request of the applicable dc_rpc_call_s function becomes free from transaction processing.

Example:

```
DCNOFLAGS|DCRPC_TPNOTRAN
```

You can specify this service request only in the transaction processing. If you specify it outside the transaction, the dc_rpc_call_s function returns a DCRPCER_INVALID_ARGS error.

(4) Arguments that contain return values

- ■ out

  The response set by the service function. This value is not returned when DCRPC_NOREPLY is specified in the flags argument.

- ■ out_len

  The length of the response set by the service function. This value is not returned when DCRPC_NOREPLY is specified in the flags argument.

(5) Return values

Return Value	Value (decimal)	Meaning
DC_OK	0	Normal termination

Return Value	Value (decimal)	Meaning
DCRPCER_INVALID_ARGS	-2401	Invalid argument
DCRPCER_PROTO	-2402	The dc_rpc_open_s function has not been executed.
DCRPCER_NO_BUFS	-2404	A sufficient amount of buffer could not be secured or resources became insufficient.
DCRPCER_NET_DOWN	-2406	Network error
DCRPCER_TIMED_OUT	-2407	The dc_rpc_call_s function processing timeout. Alternatively the service-requesting SPP aborted before completion of the processing.
DCRPCER_MESSAGE_TOO_BIG	-2408	Input parameter length exceeds the maximum.
DCRPCER_REPLY_TOO_BIG	-2409	Returned response length exceeds the area provided by the CUP.
DCRPCER_NO_SUCH_SERVICE_GROUP	-2410	Possible causes are as follows: • An undefined service group name was specified. • Although TP1/Client needs to communicate with TP1/Server, Y is specified for DCCLTNOSERVER of the client environment definition.
DCRPCER_NO_SUCH_SERVICE	-2411	An undefined service name was specified.
DCRPCER_SERVICE_CLOSED	-2412	Service group containing the specified service is shut down.
DCRPCER_SERVICE_TERMINATING	-2413	Specified service is terminating.
DCRPCER_SERVICE_NOT_UP	-2414	The SPP requested to provide a service was not started, or terminated abnormally before completing the processing. This value is returned when 0 is specified for DCWATCHTIM in the client environment definition (infinite response wait time is specified).
DCRPCER_OLTF_NOT_UP	-2415	OpenTP1 is not running on the specified service node. Alternatively, communication is impossible because the TP1/Client is disconnected from the server during a transaction.
DCRPCER_SYSERR_AT_SERVER	-2416	A system error occurred for the specified service.
DCRPCER_NO_BUFS_AT_SERVER	-2417	Insufficient memory for the specified service
DCRPCER_SYSERR	-2418	System error

Return Value	Value (decimal)	Meaning
DCRPCER_INVALID_REPLY	-2419	Response length returned to OpenTP1 from the service function is not within the range: 1 to DCRPC_MAX_MESSAGE_SIZE#.
DCRPCER_OLTF_INITIALIZING	-2420	OpenTP1 is initializing on the specified service node.
DCRPCER_NO_BUFS_RB	-2423	Insufficient memory
DCRPCER_SYSERR_RB	-2424	System error
DCRPCER_SYSERR_AT_SERVER_RB	-2425	A system error occurred for the specified service.
DCRPCER_REPLY_TOO_BIG_RB	-2426	Returned response cannot be contained in the area allocated by the CUP.
DCRPCER_TRNCHK	-2427	Transaction attribute mismatch is found among SPPs in the environment of the inter-node load balancing function. Alternatively, the version of OpenTP1 on the node used for load balancing is too old to execute the inter-node load balancing function. The return value comes only when a service request is made to the SPPs using the inter-node load balancing function.
DCRPCER_CONNFREE	-2442	The permanent connection has been released.
DCRPCER_PORT_IN_USE	-2547	The specified port number is in use, or port numbers that can be assigned automatically by the operating system are insufficient.
DCRPCER_SERVER_BUSY	-2456	The target server that receives requests from socket cannot receive a service request.
DCRPCER_TESTMODE	-2466	A service request was issued to an SPP for which test_mode=no was specified in the user service definition in an environment where DCUTOKEY was specified in the client environment definition. Alternatively, a function was called in an environment where all of the following conditions existed: • DCUTOKEY was specified in the client environment definition. • A permanent connection with the CUP executing process was being established. • The service request was issued outside the transaction. • A service request was issued to an SPP for which a value other than test_mode=no was specified in the user service definition.

Return Value	Value (decimal)	Meaning
`DCRPCER_NOT_TRN_EXTEND`	-2467	After a chained RPC has been used for transaction processing, the `dc_rpc_call_s` function that has `DCRPC_TPNORTAN` set for flags issues a service request.
`DCRPCER_SECCHK`	-2470	The service-requested SPP is protected by the security feature. The UAP that called the `dc_rpc_call_s` function has no access privilege for the server UAP.
`DCRPCER_TRNCHK_EXTEND`	-2472	Transaction branch cannot be started because the number of transaction branches that can be started concurrently has been exceeded, or, because the maximum number of child transaction branches that can be started from one transaction branch has been exceeded. Alternatively, `DCRPC_TPNOTRAN` is not set at flags in a service request qualified by a domain in a transaction.
`DCRPCER_SERVICE_TERMINATED`	-2478	The SPP requested to provide a service terminated abnormally before completing the processing. This value is returned when `00000001` is specified for `DCEXTENDFUNCTION` in the client environment definition. If `00000000` is specified or the specification is omitted, `DCRPCER_TIMED_OUT` or `DCRPCER_SERVICE_NOT_UP` returns as the return value.
`DCRPCER_VERSION_CHECK`	-2479	Since the version of service-requested TP1/Server Base is old (before 03-03), the data compression cannot be used. This return value returns when the service is requested within the range of the transaction.
`DCCLTER_INVALID_CLTID`	-2544	The client ID specified in `cltid` differs from the client ID received by the `dc_clt_cltin_s` function.
`DCRPCER_PORT_IN_USE`	-2547	The specified port number is in use. Alternatively, port numbers that can be assigned automatically by the operating system are insufficient.

# If you specify 2 or a larger value for `DCCLTRPCMAXMSGSIZE` in the client environment definition, the value you specify is used rather than the value of `DCRPC_MAX_MESSAGE_SIZE` (1 megabyte).

(6) Notes

- Do not specify the same buffer for the input parameters (`in`) and service function response (`out`).

- None of the following return values are returned if flags is set to `DCRPC_NOREPLY`.

  Errors that never occur:

  `DCRPCER_REPLY_TOO_BIG`

  `DCRPCER_INVALID_REPLY`

  Errors that cannot be detected:

  `DCRPCER_NO_SUCH_SERVICE`

  `DCRPCER_SERVICE_CLOSED`

  `DCRPCER_SERVICE_TERMINATING`

  `DCRPCER_SYSERR_AT_SERVER`

  `DCRPCER_NO_BUFS_AT_SERVER`

  `DCRPCER_OLTF_INITIALIZING`

- The possible factor for the return value `DCRPCER_TIMED_OUT` is that:

  The maximum response-wait time specified in the client environment definition is insufficient;

  The service function issued by the SPP requested to initiate service terminated abnormally;

  An error occurred with the node at which the SPP requested to initiate service exists; or

  The service-requesting SPP aborted before completion of the processing.

  A network error occurred.

  If any of the above problems occurs, the transaction started from the SPP requested to initiate service may have been committed and the database updated. Check whether the database has been updated.

- If the CUP issued the `dc_rpc_call_s` function following the `dc_trn_begin_s` function and one of the following return values was returned, then issue a rollback request function as necessary:

  `DCRPCER_TIMED_OUT`

  `DCRPCER_NO_SUCH_SERVICE`

  `DCRPCER_NO_BUFS_AT_SERVER`

  `DCRPCER_INVALID_REPLY`

  `DCRPCER_NO_BUFS_RB`

DCRPCER_SYSERR_RB

DCRPCER_SYSERR_AT_SERVER_RB

DCRPCER_REPLY_TOO_BIG_RB

4.3.4 dc_rpc_call_to_s - Request a remote service with the communication destination specified

(1) Form

(a) TP1/Client/W

■ **_s version of the function**

```
#include <dcvrpc.h>
DCLONG dc_rpc_call_to_s(
  DCCLT_ID cltid, struct DCRPC_BINDING_TBL *direction,
  char *group, char *service, char *in,
  DCULONG *in_len, char *out,
  DCULONG *out_len,
  DCLONG flags)
```

■ **Non-_s version of the function**

```
#include <dcvrpc.h>
DCLONG dc_rpc_call_to(
       struct DCRPC_BINDING_TBL *direction,
       char *group, char *service,
       char *in, DCULONG *in_len, char *out,
       DCULONG *out_len, DCLONG flags)
```

(b) TP1/Client/P

■ **_s version of the function**

```
#include <dcvrpc.h>
DCLONG dc_rpc_call_to_s(
  DCCLT_ID cltid, struct DCRPC_BINDING_TBL CLTFAR *direction,
  char CLTFAR *group, char CLTFAR *service, char CLTFAR *in,
  DCULONG CLTFAR *in_len, char CLTFAR *out,
  DCULONG CLTFAR *out_len,
  DCLONG flags)
```

■ **Non-_s version of the function**

```
#include <dcvrpc.h>
```

```
DCLONG dc_rpc_call_to(
  struct DCRPC_BINDING_TBL CLTFAR *direction, char CLTFAR
*group,
  char CLTFAR *service, char CLTFAR *in, DCULONG CLTFAR *in_len,
  char CLTFAR *out, DCULONG CLTFAR *out_len, DCLONG flags)
```

(2) Purpose

In the same way as the `dc_rpc_call_s` function, the `dc_rpc_call_to_s` function requests an SPP service. The `dc_rpc_call_to_s` function uses the host name, in addition to the service group name and service name, as a search key for the service function to restrict the nodes to which service requests are sent.

Before issuing the `dc_rpc_call_to_s` function, you must issue `DCRPC_DIRECT_SCHEDULE()` to create a `DCRPC_BINDING_TBL` structure. In the `direction` argument, specify the address of the `DCRPC_BINDING_TBL` structure. Other interfaces are the same as those for the `dc_rpc_call_s` function.

(3) Arguments set by UAPs

■ `cltid`

Specify the client ID acquired by using the `dc_clt_cltin_s` function.

■ `direction`

Specify the address of the `DCRPC_BINDING_TBL` structure.

Before issuing the `dc_rpc_call_to` or `dc_rpc_call_to_s` function, you must issue `DCRPC_DIRECT_SCHEDULE()` to specify a value in the `DCRPC_BINDING_TBL` structure.

■ `group`

Specify the service group name. The service group name can have up to 31 characters and must end with a NULL character.

■ `service`

Specify the service name. The service name can have up to 31 characters and must end with a NULL character.

■ `in`

Specify the input parameter for the service.

■ `in_len`

Specify the length of the input parameter for the service. You can specify a value in the range from 1 to `DCRPC_MAX_MESSAGE_SIZE`#.

# If you specify 2 or a larger value for `DCCLTRPCMAXMSGSIZE` in the client

environment definition, the value you specify is used rather than the value of `DCRPC_MAX_MESSAGE_SIZE` (1 megabyte).

- `out`

  Specify the address of the area for containing the service response.

- `out_len`

  Specify the length of the service response. You can specify a value in the range from 1 to `DCRPC_MAX_MESSAGE_SIZE`#.

  # If you specify 2 or a larger value for `DCCLTRPCMAXMSGSIZE` in the client environment definition, the value you specify is used rather than the value of `DCRPC_MAX_MESSAGE_SIZE` (1 megabyte).

- `flags`

  Specify the RPC mode.

  `DCNOFLAGS`

  Synchronous-response RPC

  `DCRPC_NOREPLY`

  Asynchronous-response RPC

  When `DCNOFLAGS` is specified in `flags`, the `dc_rpc_call_to_s` function does not return until the function receives a response or a timeout (specified with `DCWATCHTIM` in the client environment definition) occurs. However, if the SPP from which you requested a service aborts, the function immediately returns an error.

  In this case, the return value of this function differs depending on the timeout specified with `DCWATCHTIM`:

  - When `DCWATCHTIM` is 1 to 65535, `DCRPCER_TIMED_OUT` is returned.
  - When `DCWATCHTIM` is 0 (no timeout), `DCRPCER_SERVICE_NOT_UP` is returned.

  You can also change the timeout during execution of the CUP by executing the `dc_rpc_set_watch_time_s` function before executing the `dc_rpc_call_to_s` function.

  When `DCRPC_NOREPLY` is specified in `flags`, the function assumes that the requested service does not return a response. Therefore, the `dc_rpc_call_to_s` function immediately returns without waiting for the service to terminate. The CUP is not notified that the service function was successfully executed.

(4) Arguments that contain return values

- `out`

  The response of the service specified in the service function is returned. This value is not returned when `DCRPC_NOREPLY` is specified in the `flags` argument.

- `out_len`

  The length of the response specified in the service function is returned. This value is not returned when `DCRPC_NOREPLY` is specified in the `flags` argument.

(5) Return values

Return value	Value (decimal)	Meaning
`DC_OK`	0	Normal termination
`DCRPCER_INVALID_ARGS`	-2401	An invalid value is specified for an argument.
`DCRPCER_PROTO`	-2402	The `dc_rpc_open_s` function has not been executed. Alternatively, this function was issued while a permanent connection was being established or was issued within a transaction.
`DCRPCER_NO_BUFS`	-2404	A sufficient amount of buffer space could not be secured or resources became insufficient.
`DCRPCER_NET_DOWN`	-2406	Network error
`DCRPCER_TIMED_OUT`	-2407	A timeout occurred during processing of the `dc_rpc_call_to_s` function. Alternatively, the service-requested SPP terminated abnormally before completing the processing.
`DCRPCER_MESSAGE_TOO_BIG`	-2408	The input parameter length exceeds the maximum.
`DCRPCER_REPLY_TOO_BIG`	-2409	The length of the returned response exceeds the area provided by the CUP.
`DCRPCER_NO_SUCH_SERVICE_GROUP`	-2410	An undefined service group name was specified. Alternatively, a service request was sent to a user server that receives requests from socket (`receive_from=socket` is specified in the user service definition). Alternatively, the SPP requested to provide a service was not started when `N` was specified for `DCCLTONLYTHISNODE` of the client environment definition.
`DCRPCER_NO_SUCH_SERVICE`	-2411	An undefined service name is specified.

Return value	Value (decimal)	Meaning
DCRPCER_SERVICE_CLOSED	-2412	The service group containing the specified service is shut down.
DCRPCER_SERVICE_TERMINATING	-2413	The specified service is terminating.
DCRPCER_SERVICE_NOT_UP	-2414	The SPP requested to provide a service was not started when Y was specified for DCCLTONLYTHISNODE of the client environment definition. Alternatively, when 0 is specified for DCWATCHTIM in the client environment definition, the SPP terminated abnormally before completing the processing.
DCRPCER_OLTF_NOT_UP	-2415	OpenTP1 is not running on the node that has the specified service.
DCRPCER_SYSERR_AT_SERVER	-2416	A system error occurred for the specified service.
DCRPCER_NO_BUFS_AT_SERVER	-2417	Insufficient memory for the specified service.
DCRPCER_SYSERR	-2418	System error
DCRPCER_INVALID_REPLY	-2419	The length of the response returned to OpenTP1 from the service function is not in the range from 1 to the value of DCRPC_MAX_MESSAGE_SIZE#.
DCRPCER_OLTF_INITIALIZING	-2420	OpenTP1 is starting on the node to which the service request was sent.
DCRPCER_TRNCHK	-2427	The version of OpenTP1 on the node used for load balancing is too old to execute the inter-node load balancing facility. This value is returned only when a service request has been issued to the SPPs using the inter-node load balancing facility.
DCRPCER_TESTMODE	-2466	A service request was issued to an SPP for which test_mode=no was specified in the user service definition.
DCRPCER_SECCHK	-2470	The service-requested SPP is protected by the security facility. The UAP that called the dc_rpc_call_to_s function is not authorized to access the server UAP.

Return value	Value (decimal)	Meaning
DCRPCER_SERVICE_TERMINATED	-2478	The SPP requested to provide a service terminated abnormally before completing the processing. This value is returned when 00000001 is specified for DCEXTENDFUNCTION in the client environment definition. If 00000000 is specified or if the specification is omitted, DCRPCER_TIMED_OUT or DCRPCER_SERVICE_NOT_UP is returned as the return value.
DCCLTER_INVALID_CLTID	-2544	The client ID specified in cltid differs from the client ID received by the dc_clt_cltin_s function.
DCRPCER_PORT_IN_USE	-2547	The specified port number is in use. Alternatively, port numbers that can be assigned automatically by the operating system are insufficient.

# If you specify 2 or a larger value for DCCLTRPCMAXMSGSIZE in the client environment definition, the value you specify is used rather than the value of DCRPC_MAX_MESSAGE_SIZE (1 megabyte).

(6) Notes

- If you use the dc_rpc_call_to_s function to send a service request to a user server that receives requests from a socket (receive_from=socket is specified in the user service definition), the function returns a DCRPCER_NO_SUCH_SERVICE_GROUP error.
- The version of OpenTP1 in the destination of a service request must be 03-02 or later. The operation is not ensured if the version is earlier than 03-02.
- If the dc_rpc_call_to_s function is issued while permanent connection is being established or issued within the scope of the transaction, the function returns a DCRPCER_PROTO error.
- The values specified in DCCACHE and DCCLTCACHE in the client environment definition do not take effect.
- The dc_rpc_call_to_s function sends a request directly to the schedule service. Therefore, the value specified in DCCLTLOADBALANCE in the client environment definition does not take effect.
- When the dc_rpc_call_to_s function is issued, DCCLTSERVICEGROUPLIST in the client environment definition is not referenced.
- When the dc_rpc_call_to_s function is issued, DCSCDDIRECT, DCSCDPORT, DCSCDMULTI, and DCSCDMULTICOUNT in the client environment definition are not referenced.

- In the client environment definition, when DCCLTONLYTHISNODE is set to N or omitted, the load balancing is performed with weights assigned to the node that accepts the service request. When DCCLTONLYTHISNODE is set to Y, load balancing is not performed.
- If the host name of the node to which to send service requests is incorrect, the dc_rpc_call_to_s function returns a DCRPCER_INVALID_ARGS error.
- The behavior of the dc_rpc_call_to_s function differs depending on the value of DCCLTONLYTHISNODE in the client environment definition, as shown below.

Status of SPP		**Value of DCCLTONLYTHISNODE in the client environment definition**	
Specified node	**Other node**	**N or omitted**	**Y**
When the SPPs in these nodes have equal loads		The job is assigned to the SPP in the specified node.	The job is assigned to the SPP in the specified node.
When the SPP in the specified node has a heavier load than the SPP in the other node		The job is assigned to the SPP in the other node.	The job is assigned to the SPP in the specified node.
Active	Shut down (acceptable)	Which SPP is assigned the job depends on the load level of the SPP in the specified node. When the SPP in the specified node does not have a heavy load, the job is assigned to the SPP in the specified node. If the specified node does have a heavy load, the job is assigned to the SPP in the other node.	The job is assigned to the SPP in the specified node.
Shut down (acceptable)	Active	Which SPP is assigned the job depends on the load level of the SPP in the other node. When the SPP in the other node does not have a heavy load, the job is assigned to the SPP in the other node. If the other node does have a heavy load, the job is assigned to the SPP in the specified node.	The job is assigned to the SPP in the specified node.
Shut down (unacceptable)	Active	The job is assigned to the SPP in the other node.	The function returns a DCRPCER_SERVICE_CLOSED error.
Inactive	Active	The job is assigned to the SPP in the other node.	The function returns a DCRPCER_SERVICE_NOT_UP error.
Inactive	Inactive	The function returns a DCRPCER_NO_SUCH_SERVICE_GROUP error.	The function returns a DCRPCER_SERVICE_NOT_UP error.

4.3.5 dc_rpc_set_watch_time_s - Updating the wait time for service response

(1) Form

(a) _s version of the function

```
#include <dcvrpc.h>
DCLONG dc_rpc_set_watch_time_s(DCCLT_ID cltid, DCLONG var)
```

(b) Non-_s version of the function

```
#include <dcvrpc.h>
DCLONG dc_rpc_set_watch_time(DCLONG var)
```

(2) Purpose

Changes the timeout for the response of the service request. When the timeout is changed by using this function, the subsequent `dc_rpc_call_s` functions will use the new timeout until the `dc_rpc_close_s` function is executed. Note that this function does not change the value of `DCWATCHTIM` in the client environment definition.

Before you change the timeout by executing the `dc_rpc_set_watch_time_s` function, execute the `dc_rpc_get_watch_time_s` function to acquire the current value so that you can restore the previous setting after changing the timeout.

(3) Arguments set by UAPs

- `cltid`

  Set the client ID received by the `dc_clt_cltin_s` function.

- `var`

  Set the changed service response wait time between 1 and 65535. Specifying `0` provides an infinite wait condition.

(4) Return values

Return Value	Value (decimal)	Meaning
`DC_OK`	0	Normal termination
`DCRPCER_INVALID_ARGS`	-2401	An invalid value is specified for var.
`DCRPCER_PROTO`	-2402	The `dc_rpc_open_s` function is not executed.
`DCRPCER_NO_BUFS`	-2404	Insufficient memory

Return Value	Value (decimal)	Meaning
DCRPCER_INVALID_CLTID	-2544	The client ID specified for cltid differs from the one received from the dc_clt_cltin_s function.

4.3.6 dc_rpc_get_watch_time_s - Referencing the wait time for service response

(1) Form

(a) _s version of the function

```
#include <dcvrpc.h>
DCLONG dc_rpc_get_watch_time_s(DCCLT_ID cltid)
```

(b) Non-_s version of the function

```
#include <dcvrpc.h>
DCLONG dc_rpc_get_watch_time()
```

(2) Purpose

References the response wait time for the current service request.

You can use this function to save the original value before temporarily changing the service response wait time using the dc_rpc_set_watch_time_s function.

The dc_rpc_get_watch_time_s function returns the service response wait time changed with the dc_rpc_set_watch_time_s function. When the wait time remains unchanged, the function returns the DCWATCHTIM value in the client environment definition.

Values obtained by the dc_rpc_get_watch_time_s function are available for the dc_rpc_call_s function.

(3) Argument set by UAPs

- cltid

  Set the client ID received by the dc_clt_cltin_s function.

(4) Return values

Return Value	Value (decimal)	Meaning
-	Positive integer	Current service response wait time
-	0	The service response wait time is indefinite.

Return Value	Value (decimal)	Meaning
`DCRPCER_PROTO`	-2402	The `dc_rpc_open_s` function is not executed.
`DCRPCER_NO_BUFS`	-2404	Insufficient memory
`DCRPCER_INVALID_CLTID`	-2544	The client ID specified for `cltid` differs from the one received from the `dc_clt_cltin_s` function.

Legend:

-: Not applicable

4.3.7 DCRPC_DIRECT_SCHEDULE - Create a DCRPC_BINDING_TBL structure

(1) Form

(a) TP1/Client/W

```
#include <dcvrpc.h>
DCRPC_DIRECT_SCHEDULE(
             struct DCRPC_BINDING_TBL *direction,
             char *hostnm,
             unsigned short scdport, DCLONG flags)
```

(b) TP1/Client/P

```
#include <dcvrpc.h>
DCRPC_DIRECT_SCHEDULE(
      struct DCRPC_BINDING_TBL CLTFAR *direction,
      char CLTFAR *hostnm, unsigned short scdport,
      DCLONG flags)
```

(2) Purpose

The `DCRPC_DIRECT_SCHEDULE` function creates the `DCRPC_BINDING_TBL` structure to be specified in an argument of the `dc_rpc_call_to_s` function.

(3) Arguments set by UAPs

- `direction`

  Specify the address of the `DCRPC_BINDING_TBL` structure. Create a `DCRPC_BINDING_TBL` structure for each thread.

- `hostnm`

  Specify the address of the area containing the name of the host to which you want to send service requests. End the address with a NULL character.

If you set NULL, the subsequent `dc_rpc_call_to_s` function returns a `DCRPCER_INVALID_ARGS` error.

You can specify a maximum of 63# characters for the host name.

As the host name, you can also specify an IP address in the dotted decimal format.

# If you specify `00000008` for `DCCLTOPTION` in the client environment definition, you can specify a maximum of 255 characters for the host name.

- `scdport`

  Specify the port number of the schedule service existing in the host to which you want to send service requests. The port number is specified in the `scd_port` clause in the schedule service definition. As the port number, you can set 0 or another value in the range from 5001 to 65535.

  If you specify `0`, the subsequent `dc_rpc_call_to_s` function queries the server about the name service that authenticates the user.

- `flags`

  Set `DCNOFLAGS`.

4.4 Permanent connection

4.4.1 dc_clt_connect_s - Establish permanent connection

(1) Form

(a) _s version of the function

```
#include <dcvclt.h>
DCLONG dc_clt_connect_s(DCCLT_ID cltid, DCLONG flags)
```

(b) Non-_s version of the function

```
#include <dcvclt.h>
DCLONG dc_clt_connect(DCLONG flags)
```

(2) Purpose

Establishes permanent connection with a CUP execution process, a RAP-processing server or DCCM3 logical terminal.

The CUP execution process for establishing the permanent connection is running on the OpenTP1 node specified in the `target_host` in the `dc_clt_cltin_s` function, specified in `DCCLTRAPHOST` or `DCHOST` in the client environment definition.

To establish the permanent connection with the DCCM3 logical terminal, define `DCCLTDCCMHOST` and `DCCLTDCCMPORT` in the client environment definition. Also specify `DCCLT_DCCM3` for the argument flags in the `dc_clt_connect_s` function.

When you establish permanent connection with the DCCM3 logical terminal using the remote API facility, provide `DCCLTRAPHOST` with the host name and the port number for the DCCM3 logical terminal. Specify `DCNOFLAGS` for flags of the `dc_clt_connect_s` function.

(3) Arguments set by UAPs

- `cltid`

  Specify the client ID received by the `dc_clt_cltin_s` function.

- `flags`

  Specify either of the following to establish permanent connection.

  `DCNOFLAGS`

  Permanent connection is established with TP1/Server, a RAP-processing server or DCCM3 logical terminal.

`DCCLT_DCCM3`

Permanent connection is established with a DCCM3 logical terminal.

(4) Return values

Return Value	Value (decimal)	Meaning
`DC_OK`	0	Normal termination. Or, permanent connection has already been established.
`DCCLTER_INVALID_ARGS`	-2501	Invalid argument
`DCCLTER_PROTO`	-2502	The `dc_clt_connect` or `dc_clt_connect_s` function is issued in the transaction, or the `dc_rpc_open_s` function is not issued. The establishment request to OpenTP1 is issued while permanent connection with DCCM3 has already been established. Or, the establishment request to DCCM3 is issued while permanent connection with OpenTP1 has already been established.
`DCCLTER_NO_BUFS`	-2504	A necessary amount of buffer could not be allocated. Alternatively, resources became insufficient.
`DCCLTER_NET_DOWN`	-2506	Communication error
`DCCLTER_TIMED_OUT`	-2507	A timeout error occurred during establishment of permanent connection.
`DCCLTER_OLTF_NOT_UP`	-2515	One of the following causes is likely: • The OpenTP1 server or the DCCM3 logical terminal has not started. • The client extended service has not started. `clt_conf` is specified incorrectly in the system service configuration definition. • The CUP executing process has not started. `clt_cup_conf` is specified incorrectly in the client service definition.
`DCCLTER_SYSERR`	-2518	System error
`DCCLTER_WRONG_HOST`	-2539	The establishment request to the DCCM3 logical terminal is issued with an invalid host name.
`DCCLTER_INVALID_CLTID`	-2544	The client ID specified in `cltid` differs from the client ID received by the `dc_clt_cltin_s` function.
`DCCLTER_PORT_IN_USE`	-2547	The specified port number is in use, or port numbers that can be assigned automatically by the operating system are insufficient.

(5) Notes

- No permanent connection is established when the `dc_clt_connect_s` function returns error. Permanent connection may be established only on the CUP execution process or DCCM3 logical terminal if the return value is `DCCLTER_NET_DOWN`, `DCCLTER_TIMED_OUT`, or `DCCLTER_SYSERR`.

  In this case, the CUP execution process or DCCM3 logical terminal may keep on waiting for a response from the CUP. To prevent an infinite wait, specify an appropriate value for the maximum time interval for the permanent connection. For a DCCM3 logical terminal, specify an appropriate value for the time during which the system is unable to determine whether a connection with the terminal is valid.
- The `dc_clt_connect_s` function cannot be issued in a transaction.
- You can establish permanent connection with only one of the following two categories.
  - CUP execution process, RAP-processing server, or a DCCM3 logical terminal that is specified for `DCCLTRAPHOST` in the client environment definition
  - DCCM3 logical terminal that is specified for `DCCLTDCCMHOST` in the client environment definition

    If you establish permanent connection with one category, you cannot communicate with the other until you issue the `dc_clt_disconnect_s` function.
- The data compression is unavailable when you establish permanent connection with DCCM3 logical terminals. You need to omit `DCCLTDATACOMP` or specify `N` for it in the client environment definition.

4.4.2 dc_clt_disconnect_s - Release permanent connection

(1) Form

(a) _s version of the function

```
#include <dcvclt.h>
DCLONG dc_clt_disconnect_s(DCCLT_ID cltid, DCLONG flags)
```

(b) Non-_s version of the function

```
#include <dcvclt.h>
DCLONG dc_clt_disconnect(DCLONG flags)
```

(2) Purpose

Releases permanent connection with a CUP execution process, a RAP-processing server or DCCM3 logical terminal.

(3) Arguments set by UAPs

- `cltid`

  Specify the client ID received by the `dc_clt_cltin_s` function.

- `flags`

  Set `DCNOFLAGS`.

(4) Return values

Return Value	Value (decimal)	Meaning
`DC_OK`	0	Normal termination. Alternatively, for TP1/Client/P, the permanent connection is already disconnected.
`DCCLTER_INVALID_ARGS`	-2501	Invalid argument
`DCCLTER_PROTO`	-2502	The `dc_rpc_open_s` function is not issued.
`DCCLTER_NO_BUFS`	-2504	A necessary amount of buffer could not be allocated.
`DCCLTER_NET_DOWN`	-2506	Communication error. Alternatively, for TP1/Client/P, the permanent connection is already disconnected.
`DCCLTER_TIMED_OUT`	-2507	A timeout error occurred during release of permanent connection.
`DCCLTER_SYSERR`	-2518	System error.
`DCCLTER_INVALID_CLTID`	-2544	The client ID specified in `cltid` differs from the client ID received by the `dc_clt_cltin_s` function.

(5) Notes

- The permanent connection is not released when the `dc_clt_disconnect_s` function returns an error with one the following return values:
  - `DCCLTER_INVALID_ARGS`
  - `DCCLTER_PROTO`
  - `DCCLTER_NO_BUFS` (when the error is detected on the client)
  - `DCCLTER_INVALID_CLTID`
- TP1/Client forcibly releases the permanent connection if the `dc_clt_disconnect_s` function returns an error with one of the following

return values:

- `DCCLTER_NO_BUFS` (when the error is detected on the server)
- `DCCLTER_NET_DOWN`
- `DCCLTER_TIMED_OUT`
- `DCCLTER_SYSERR`

In this case, the CUP execution process or DCCM3 logical terminal may keep on waiting for a response from the CUP, without detecting the release of permanent connection by TP1/Client. To prevent an infinite wait, specify an appropriate value for the maximum time interval for the permanent connection. For a DCCM3 logical terminal, specify an appropriate value for the time during which the system is unable to determine whether a connection with the terminal is valid.

- Issuing the `dc_clt_disconnect_s` function in a transaction commits the transaction.

4.4.3 dc_clt_set_raphost_s - Set the destination of a request to establish a permanent connection

(1) Form

(a) TP1/Client/W

■ **_s version of the function**

```
#include <dcvclt.h>
DCLONG dc_clt_set_raphost_s(DCCLT_ID cltid,
                            char *raphost,
                            DCLONG flags)
```

■ **Non-_s version of the function**

```
#include <dcvclt.h>
DCLONG dc_clt_set_raphost(char *raphost, DCLONG flags)
```

(b) TP1/Client/P

■ **_s version of the function**

```
#include <dcvclt.h>
DCLONG dc_clt_set_raphost_s(DCCLT_ID cltid,
                            char CLTFAR *raphost,
                            DCLONG flags)
```

■ Non-_s version of the function

```
#include <dcvclt.h>
DCLONG dc_clt_set_raphost(char CLTFAR *raphost, DCLONG flags)
```

(2) Purpose

The `dc_clt_set_raphost_s` function sets the host name and port number of the node to which you want to send a request to establish a permanent connection. The host name and port number set by these functions prevail over those specified in `DCCLTRAPHOST` in the client environment definition. After the `dc_clt_set_raphost_s` function is executed, the `dc_clt_connect_s` function uses the host name and port number specified in the `dc_clt_set_raphost_s` function.

You may want to restore the host name and port number that were used before the `dc_clt_set_raphost_s` function was executed. To do this, before executing the `dc_clt_set_raphost_s` function to set a new host name and port number, execute the `dc_clt_get_raphost_s` function to acquire the current host name and port number. Then, after executing the `dc_clt_set_raphost_s` function to set a new host name and port number, reexecute the function specifying the previously acquired host name and port number.

(3) Arguments set by UAPs

■ `cltid`

Specify the client ID received by the `dc_clt_cltin_s` function.

■ `raphost`

Specify a pointer to an area of at least 256 bytes# where the host name and port number of the host to which a request for establishing a permanent connection is to be sent is set.

#:

When `00000008` is specified for `DCCLTOPTION` of the client environment definition, the minimum size of the area is 1024 bytes, not 256 bytes.

■ `flags`

Specify `DCNOFLAGS`.

(4) Return values

Return value	Value (decimal)	Meaning
`DC_OK`	0	The function normally terminated.

Return value	Value (decimal)	Meaning
DCCLTER_INVALID_ARGS	-2501	The value specified in an argument is incorrect.
DCCLTER_PROTO	-2502	The function has already been issued in the transaction or is now establishing a permanent connection. Alternatively, the dc_rpc_open_s function has not been issued.
DCCLTER_NO_BUFS	-2504	A necessary amount of buffer could not be allocated.
DCCLTER_INVALID_CLTID	-2544	The client ID specified in cltid differs from the client ID acquired by the dc_clt_cltin_s function.

(5) Notes

- The dc_clt_set_raphost_s functions do not change the value of DCCLTRAPHOST in the client environment definition.
- If raphost specifies a pointer to a NULL character, DCCLTRAPHOST is placed in undefined status. When DCCLTRAPHOST is not defined, the dc_clt_connect_s function establishes a permanent connection to the logical terminal of the CUP executing process or of DCCM3.

4.4.4 dc_clt_get_raphost_s - Acquire the destination of a request to establish a permanent connection

(1) Form

(a) TP1/Client/W

■ **_s version of the function**

```
#include <dcvclt.h>
DCLONG dc_clt_get_raphost_s(DCCLT_ID cltid,
                            char *raphost,
                            DCLONG flags)
```

■ **Non-_s version of the function**

```
#include <dcvclt.h>
DCLONG dc_clt_get_raphost(char *raphost, DCLONG flags)
```

(b) TP1/Client/P

■ **_s version of the function**

```
#include <dcvclt.h>
```

```
DCLONG dc_clt_get_raphost_s(DCCLT_ID cltid,
                            char CLTFAR *raphost,
                            DCLONG flags)
```

■ Non-_s version of the function

```
#include <dcvclt.h>
DCLONG dc_clt_get_raphost(char CLTFAR *raphost, DCLONG flags)
```

(2) Purpose

The `dc_clt_get_raphost_s` function acquires the host name and port number of the node to which to send a request to establish a permanent connection.

Before executing the `dc_clt_set_raphost_s` function to specify the new destination of a request to establish a permanent connection, execute the `dc_clt_get_raphost_s` function to save the current destination.

When the `dc_clt_get_raphost_s` function is executed, the latest destination set by the `dc_clt_set_raphost_s` function is returned to `raphost`. If the `dc_clt_set_raphost_s` function has not been executed, the value of `DCCLTRAPHOST` in the client environment definition is returned to `raphost`.

(3) Arguments set by UAPs

■ `cltid`

Specify the client ID received by the `dc_clt_cltin_s` function.

■ `flags`

Specify `DCNOFLAGS`.

■ `raphost`

Specify a pointer to the area larger than 256 bytes# for containing the host name and port number of the current destination of a request to establish a permanent connection.

# If you specify `00000008` for `DCCLTOPTION` in the client environment definition, this value is 1,024 bytes, not 256 bytes.

(4) Arguments that contain return values

■ `raphost`

The current host name and port number of the destination of a request to establish a permanent connection are returned to `raphost`. If `DCCLTRAPHOST` in the client environment definition is not defined and the destination is not set by the `dc_clt_set_raphost_s` function, a NULL character is returned to the beginning of `raphost`.

Form:

host-name[:*port-number*][,*host-name*[:*port-number*],...]

host-name ~<character string>

The host name of the destination of a request to establish a permanent connection is returned.

port-number ~<unsigned integer>((5001 to 65535))

The port number of the destination of a request to establish a permanent connection is returned.

(5) Return values

Return value	Value (decimal)	Meaning
`DC_OK`	0	The function normally terminated.
`DCCLTER_INVALID_ARGS`	-2501	The value specified in an argument is incorrect.
`DCCLTER_PROTO`	-2502	The `dc_rpc_open_s` function has not been issued.
`DCCLTER_NO_BUFS`	-2504	A necessary amount of buffer could not be allocated.
`DCCLTER_INVALID_CLTID`	-2544	The client ID specified in `cltid` differs from the client ID acquired by the `dc_clt_cltin_s` function.
`DCCLTER_DLL_NOT_LOADED`	-2555	The specified DLL could not be loaded.
`DCCLTER_FUNC_NOT_DEFINED`	-2556	An attempt was made to issue a function not defined in the specified DLL.

(6) Note

Specify an area of 256 bytes# or more for the `raphost` argument. If the area is smaller than 256 bytes#, the area may be corrupted during TP1/Client internal processing.

#:

If you specify `00000008` for `DCCLTOPTION` in the client environment definition, this value is 1,024 bytes, not 256 bytes.

4.4.5 dc_clt_set_connect_inf_s - Set terminal identification information

(1) Form

(a) TP1/Client/W

■ **_s version of the function**

```
#include <dcvclt.h>
DCLONG dc_clt_set_connect_inf_s(DCCLT_ID cltid,
                                char *inf,
                                unsigned short inf_len,
                                DCLONG flags)
```

■ **Non-_s version of the function**

```
#include <dcvclt.h>
DCLONG dc_clt_set_connect_inf(char *inf,
                              unsigned short inf_len,
                              DCLONG flags)
```

(b) TP1/Client/P

■ **_s version of the function**

```
#include <dcvclt.h>
DCLONG dc_clt_set_connect_inf_s(DCCLT_ID cltid,
                                char CLTFAR *inf,
                                unsigned short inf_len,
                                DCLONG flags)
```

■ **Non-_s version of the function**

```
#include <dcvclt.h>
DCLONG dc_clt_set_connect_inf(char CLTFAR *inf,
                              unsigned short inf_len,
                              DCLONG flags)
```

(2) Purpose

Sets terminal identification information dynamically.

When you use a permanent connection to communicate with a DCCM3 logical terminal, reporting the terminal identification information to the DCCM3 logical terminal allows you to use DCCM3's function for allocating a fixed terminal.

The terminal identification information specified in this function is valid only when the host name and the port number of the DCCM3 logical terminal are specified in the DCCLTRAPHOST client environment definition and DCNOFLAGS is specified in the flags argument of the dc_clt_connect_s function.

When this function is executed, the terminal identification information specified in the DCCLTCONNECTINF client environment definition is not referenced until the the dc_rpc_open_s function is reexecuted.

The terminal identification information specified in this function is referenced by the dc_clt_connect_s function which is executed after this function. These functions then report that information to the DCCM3 logical terminal.

If this function is executed more than once, the dc_clt_connect_s function references the terminal identification information specified immediately before the function.

(3) Arguments set by UAPs

- cltid

  Specify the client ID received by the dc_clt_cltin_s function.

- inf

  Specify the terminal identification information. If you want to use hexadecimal numbers to specify this information, use up to 64 bytes. If you want to use a character string to specify this information, use up to 64 characters (excluding any NULL characters).

  When you use a permanent connection to communicate with a DCCM3 logical terminal, use EBCDIK code to specify the logical terminal name of the DCCM3 logical terminal as the terminal identification information. However, DCCM3 only validates the first 8 bytes (the 9th and later bytes are ignored).

- inf_len

  Specify the terminal identification information length. You can specify a length between 1 and DCCLT_MAX_CONNECT_INF_SIZE. DCCLT_MAX_CONNECT_INF_SIZE is defined in the header file. The header file is dcvclt.h for TP1/Client/W. For TP1/Client/P, the header file is DCVCLT.H.

- flags

  Specify DCNOFLAGS.

(4) Return values

Return Value	Value (decimal)	Meaning
DC_OK	0	Normal termination

Return Value	Value (decimal)	Meaning
`DCCLTER_INVALID_ARGS`	-2501	The value specified as the argument is incorrect.
`DCCLTER_PROTO`	-2502	The `dc_rpc_open_s` function is not executed.
`DCCLTER_NO_BUFS`	-2504	A necessary amount of buffer could not be allocated.
`DCCLTER_INVALID_CLTID`	-2544	The client ID specified in `cltid` differs from the client ID received by the `dc_clt_cltin_s` function.

(5) Notes

- Reporting terminal identification information allows you to use DCCM3' function for allocating a fixed terminal only when DCCM3 is version 09-03 or later. For details about the function for allocating a fixed terminal, see the manual *VOS3 Data Management System XDM E2 Description*.
- If the logical terminal name of the DCCM3 logical terminal matching the terminal identification information defined in the `dc_clt_set_connect_inf_s` function is not defined in DCCM3, the `dc_clt_connect_s` function returns a `DCCLTER_NET_DOWN` error.

4.5 Transaction control

4.5.1 dc_trn_begin_s - Transaction startup

(1) Form

(a) _s version of the function

```
#include <dcvtrn.h>
DCLONG dc_trn_begin_s(DCCLT_ID cltid)
```

(b) Non-_s version of the function

```
#include <dcvtrn.h>
DCLONG dc_trn_begin()
```

(2) Purpose

Starts a global transaction from the CUP process that issues the `dc_trn_begin_s` function.

The `dc_trn_begin_s` function must be issued after the `dc_rpc_open_s` function.

One global transaction covers from issuing the `dc_trn_begin_s` function to a synchronous point (commit request) of the transaction. In the global transaction, the `dc_trn_begin_s` function cannot be duplicated (that of an SPP included). Duplication of the function causes an error return.

The SPP transaction attribute follows the `atomic_update` specification of the user service definition.

(3) Arguments set by UAPs

- `cltid`

  Set the client ID received by the `dc_clt_cltin_s` function.

(4) Return values

Return Value	Value (decimal)	Meaning
`DC_OK`	0	Normal termination

Return Value	Value (decimal)	Meaning
`DCCLTER_PROTO`	-2502	The function has been issued from an invalid context (for example, from within a transaction). Alternatively, the function has been issued from an environment where both of the following conditions exist: • `DCUTOKEY` is specified in the client environment definition. • A permanent connection is being established with a RAP-processing server.
`DCCLTER_NO_BUFS`	-2504	Insufficient memory. Alternatively, the resource became insufficient.
`DCCLTER_NET_DOWN`	-2506	Network error
`DCCLTER_TIMED_OUT`	-2507	Timeout occurred during the `dc_trn_begin_s` function processing.
`DCCLTER_NO_SUCH_SERVICE_GROUP`	-2510	The client extended service has not started. Check whether `clt_conf` is specified correctly in the system service configuration definition. Alternatively, the transactional RPC executing process has not started. Check whether `clt_trn_conf` is specified correctly in the client service definition.
`DCCLTER_OLTF_NOT_UP`	-2515	OpenTP1 has not been activated.
`DCCLTER_NO_BUFS_AT_SERVER`	-2517	Memory became insufficient in a transaction process.
`DCCLTER_SYSERR`	-2518	System error
`DCCLTER_CONNFREE`	-2542	The permanent connection has been released.
`DCCLTER_INVALID_CLTID`	-2544	The client ID specified for `cltid` differs from the one received from the `dc_clt_cltin_s` function.
`DCCLTER_BUSY_AT_SERVER`	-2545	Transaction cannot occur because of an excessive load on a transaction process on the server. Reexecute the transaction, which would be successful, when receiving this return value.
`DCCLTER_PORT_IN_USE`	-2547	The specified port number is in use, or port numbers that can be assigned automatically by the operating system are insufficient.
`DCTRNER_RM`	-3406	An error occurred in the Resource Manager (RM). No transaction could occur.

Return Value	Value (decimal)	Meaning
DCTRNER_TM	-3407	No transaction could occur because error was generated in the transaction service. Reexecute the transaction, which would be successful, when receiving this return value.

4.5.2 dc_trn_chained_commit_s - Commit in chained mode

(1) Form

(a) _s version of the function

```
#include <dcvtrn.h>
DCLONG dc_trn_chained_commit_s(DCCLT_ID cltid)
```

(b) Non-_s version of the function

```
#include <dcvtrn.h>
DCLONG dc_trn_chained_commit()
```

(2) Purpose

Acquires the synchronous point of a transaction.

When the `dc_trn_chained_commit_s` function terminates normally, a new global transaction occurs. All functions that follow fall in the range of the new global transaction.

(3) Arguments set by UAPs

- `cltid`

  Set the client ID received by the `dc_clt_cltin_s` function.

(4) Return values

Return Value	Value (decimal)	Meaning
DC_OK	0	Normal termination
DCCLTER_PROTO	-2502	The function has been issued from an invalid context (for example, from outside a transaction).
DCCLTER_NO_BUFS	-2504	Insufficient memory
DCCLTER_NET_DOWN	-2506	Network error

Return Value	Value (decimal)	Meaning
`DCCLTER_TIMED_OUT`	-2507	A timeout error occurred in the processing of the `dc_trn_chained_commit_s` function.
`DCCLTER_OLTF_NOT_UP`	-2515	OpenTP1 has not been activated. Alternatively, communication is impossible because the TP1/Client is disconnected from the server.
`DCCLTER_NO_BUFS_AT_SERVER`	-2517	Memory became insufficient in a transaction process.
`DCCLTER_SYSERR`	-2518	System error
`DCCLTER_CONNFREE`	-2542	The permanent connection has been released.
`DCCLTER_INVALID_CLTID`	-2544	The client ID specified for `cltid` differs from the one received by the `dc_clt_cltin_s` function.
`DCTRNER_ROLLBACK`	-3402	Current transaction failed to be committed and was rolled back. After completion of the rollback, the process will be under the transaction and in the global transaction.
`DCTRNER_HEURISTIC`	-3403	One transaction branch was committed and another was rolled back with heuristic determination. This return value will be returned if the result of the heuristic determination does not match the one of the synchronous point of this global transaction. For information about causes of this return value and the result of the synchronous point, see the message log file. After this return value is returned, the process is still under the transaction and in the global transaction.
`DCTRNER_HAZARD`	-3404	A transaction branch of the global transaction terminated heuristically. However, the result of the synchronous point of this transaction branch was not apparent because of the error. For information about causes of this return value and the result of the synchronous point, see the message log file. After this return value is returned, the process is still under the transaction and in the global transaction.
`DCTRNER_NO_BEGIN`	-3424	Current transaction is committed and terminated normally. But a new transaction could not start. After this return value is returned, the process will not under the transaction.
`DCTRNER_ROLLBACK_NO_BEGIN`	-3425	Current transaction could not be committed and was rolled back. Further new transactions could not start. The process is not under the transaction.

Return Value	Value (decimal)	Meaning
`DCTRNER_HEURISTIC_NO_BEGIN`	-3426	The global transaction that executed the `dc_trn_chained_commit_s` function follows the heuristic determination. Some transactions may or may not be committed. This return value will be returned if the result of the heuristic determination differs from the result of the synchronous point for the global transaction. For the result of the synchronous point for the UAP, resource manager, or global transaction that caused this return value, see the message log file. Further new transactions could not start. The process is not under the transaction.
`DCTRNER_HAZARD_NO_BEGIN`	-3427	The global transaction's transaction branch has completed heuristically. But an error makes it impossible to determine the result of the synchronous point for this transaction branch. For the result of the synchronous point for the UAP, resource manager, or global transaction that caused this return value, see the message log file. Further new transactions could not start. The process is not under the transaction.

(5) Notes

To terminate the CUP process after committing the transaction, always execute the `dc_trn_unchained_commit_s` function.

4.5.3 dc_trn_chained_rollback_s - Rollback in chained mode

(1) Form

(a) _s version of the function

```
#include <dcvtrn.h>
DCLONG dc_trn_chained_rollback_s(DCCLT_ID cltid)
```

(b) Non-_s version of the function

```
#include <dcvtrn.h>
DCLONG dc_trn_chained_rollback()
```

(2) Purpose

Rolls back a transaction.

When the `dc_trn_chained_rollback_s` function terminates normally, a new global transaction occurs. All functions that follow fall in the range of the new global transaction.

(3) Arguments set by UAPs

- `cltid`

  Set the client ID received by the `dc_clt_cltin_s` function.

(4) Return values

Return Value	Value (decimal)	Meaning
`DC_OK`	0	Normal termination
`DCCLTER_PROTO`	-2502	The function has been issued from an invalid context (for example, from outside a transaction).
`DCCLTER_NO_BUFS`	-2504	Insufficient memory
`DCCLTER_NET_DOWN`	-2506	Network error
`DCCLTER_TIMED_OUT`	-2507	Timeout occurred during the `dc_trn_chained_rollback_s` function processing.
`DCCLTER_OLTF_NOT_UP`	-2515	OpenTP1 has not been activated. Alternatively, communication is impossible because the TP1/Client is disconnected from the server.
`DCCLTER_NO_BUFS_AT_SERVER`	-2517	Memory became insufficient in a transaction process.
`DCCLTER_SYSERR`	-2518	System error
`DCCLTER_CONNFREE`	-2542	The permanent connection has been released.
`DCCLTER_INVALID_CLTID`	-2544	The client ID specified for `cltid` differs from the one received by the `dc_clt_cltin_s` function.
`DCTRNER_HEURISTIC`	-3403	One transaction branch was committed and another was rolled back with heuristic determination. This return value will be returned if the result of the heuristic determination does not match the one of the synchronous point of this global transaction. For information about causes of this return value and the result of the synchronous point, see the message log file. After this return value is returned, the process is still under the transaction and in the global transaction.

Return Value	Value (decimal)	Meaning
`DCTRNER_HAZARD`	-3404	Transaction branch of the global transaction terminated heuristically. However, the result of synchronous point of this transaction branch was not apparent because of the error. For information about causes of the return value and the result of the synchronous point, see the message log file. After this return value is returned, the process is still under the transaction and in the global transaction.
`DCTRNER_NO_BEGIN`	-3424	Current transaction rolled back normally. But a new transaction could not start. After this return value is returned, the process will not under the transaction.
`DCTRNER_HEURISTIC_NO_BEGIN`	-3426	The global transaction that executed the `dc_trn_chained_rollback_s` function follows the heuristic determination. Some transactions may or may not be committed. This return value will be returned if the result of the heuristic determination differs from the result of the synchronous point for the global transaction. For the result of the synchronous point for the UAP, resource manager, or global transaction that caused this return value, see the message log file. Further new transactions could not start. The process is not under the transaction.
`DCTRNER_HAZARD_NO_BEGIN`	-3427	The global transaction's transaction branch has completed heuristically. But an error makes it impossible to determine the result of the synchronous point for this transaction branch. For the result of the synchronous point for the UAP, resource manager, or global transaction that caused this return value, see the message log file. Further new transactions could not start. The process is not under the transaction.

(5) Notes

To terminate a CUP process after rolling back the transaction, be sure to execute the `dc_trn_unchained_rollback_s` function.

4.5.4 dc_trn_unchained_commit_s - Commit in unchained mode

(1) Form

(a) _s version of the function

```
#include <dcvtrn.h>
DCLONG dc_trn_unchained_commit_s(DCCLT_ID cltid)
```

(b) Non-_s version of the function

```
#include <dcvtrn.h>
DCLONG dc_trn_unchained_commit()
```

(2) Purpose

Acquires the synchronous point of a transaction.

When the `dc_trn_unchained_commit_s` function terminates normally, the global transaction also terminates. No SPP can be executed as a transaction from outside the global transaction.

(3) Arguments set by UAPs

- `cltid`

  Set the client ID received by the `dc_clt_cltin_s` function.

(4) Return values

Return Value	Value (decimal)	Meaning
`DC_OK`	0	Normal termination
`DCCLTER_PROTO`	-2502	The function has been issued from an invalid context (for example, from outside a transaction).
`DCCLTER_NO_BUFS`	-2504	Insufficient memory
`DCCLTER_NET_DOWN`	-2506	Network error
`DCCLTER_TIMED_OUT`	-2507	Timeout occurred during the `dc_trn_unchained_commit_s` function processing.
`DCCLTER_OLTF_NOT_UP`	-2515	OpenTP1 has not been activated. Alternatively, communication is impossible because the TP1/Client is disconnected from the server.
`DCCLTER_NO_BUFS_AT_SERVER`	-2517	Memory became insufficient in a transaction process.
`DCCLTER_SYSERR`	-2518	System error
`DCCLTER_CONNFREE`	-2542	The permanent connection has been released.
`DCCLTER_INVALID_CLTID`	-2544	The client ID specified for `cltid` differs from the one received by the `dc_clt_cltin_s` function.

Return Value	Value (decimal)	Meaning
`DCTRNER_ROLLBACK`	-3402	Transaction was rolled back because it failed to be committed. After this return value is returned, the process will be outside the global transaction.
`DCTRNER_HEURISTIC`	-3403	Some or all transaction branches were rolled back with heuristic determination. See details on the message log file. After this return value is returned, the process is outside the global transaction.
`DCTRNER_HAZARD`	-3404	Transaction terminated with heuristic determination, but its result was not apparent because of the error. See details on the message log file. After this return value is returned, the process is outside the global transaction.

(5) Notes

To terminate a CUP process normally, issue the `dc_trn_unchained_commit_s` function to commit the transaction.

4.5.5 dc_trn_unchained_rollback_s - Rollback in unchained mode

(1) Form

(a) _s version of the function

```
#include <dcvtrn.h>
DCLONG dc_trn_unchained_rollback_s(DCCLT_ID cltid)
```

(b) Non-_s version of the function

```
#include <dcvtrn.h>
DCLONG dc_trn_unchained_rollback()
```

(2) Purpose

Rolls back a transaction.

When the `dc_trn_unchained_rollback_s` function terminates normally, the global transaction terminates. No SPP can be executed as a transaction from outside the global transaction.

(3) Arguments set by UAPs

- `cltid`

  Set the client ID received by the `dc_clt_cltin_s` function.

(4) Return values

Return Value	Value (decimal)	Meaning
`DC_OK`	0	Normal termination
`DCCLTER_PROTO`	-2502	The function has been issued from an invalid context (for example, from outside a transaction).
`DCCLTER_NO_BUFS`	-2504	Insufficient memory
`DCCLTER_NET_DOWN`	-2506	Network error
`DCCLTER_TIMED_OUT`	-2507	Timeout occurred during the `dc_trn_unchained_rollback_s` function processing.
`DCCLTER_OLTF_NOT_UP`	-2515	OpenTP1 has not been activated. Alternatively, communication is impossible because the TP1/Client is disconnected from the server.
`DCCLTER_NO_BUFS_AT_SERVER`	-2517	Memory became insufficient in a transaction process.
`DCCLTER_SYSERR`	-2518	System error
`DCCLTER_CONNFREE`	-2542	The permanent connection has been released.
`DCCLTER_INVALID_CLTID`	-2544	The client ID specified for `cltid` differs from the one received by the `dc_clt_cltin_s` function.
`DCTRNER_HEURISTIC`	-3403	Some or all transaction branches were rollbacked with heuristic determination. See details on the message log file. After this return value is returned, the process is outside the global transaction.
`DCTRNER_HAZARD`	-3404	Transaction terminated with heuristic determination, but its result was not apparent because of the error. See details on the message log file. After this return value is returned, the process is outside the global transaction.

(5) Notes

To terminate a CUP process normally after the transaction has been rolled back, issue the `dc_trn_unchained_rollback_s` function.

4.5.6 dc_clt_get_trnid_s - Collection of identifiers for current transaction

(1) Form

(a) TP1/Client/W

■ **_s version of the function**

```
#include <dcvclt.h>
DCLONG dc_clt_get_trnid_s(DCCLT_ID cltid, char *trngid,
                          char *trnbid)
```

■ **Non-_s version of the function**

```
#include <dcvclt.h>
DCLONG dc_clt_get_trnid(char *trngid, char *trnbid)
```

(b) TP1/Client/P

■ **_s version of the function**

```
#include <dcvclt.h>
DCLONG dc_clt_get_trnid_s(DCCLT_ID cltid,
                          char CLTFAR *trngid,
                          char CLTFAR *trnbid)
```

■ **Non-_s version of the function**

```
#include <dcvclt.h>
DCLONG dc_clt_get_trnid(char CLTFAR *trngid, char CLTFAR
*trnbid)
```

(2) Purpose

Collects the global identifier for the current transaction and the identifier for the current transaction branch.

These identifiers were assigned by OpenTP1 when the following functions were issued to start the transaction:

- `dc_trn_begin_s`
- `dc_trn_chained_commit_s`
- `dc_trn_chained_rollback_s`

(3) Arguments set by UAPs

- `cltid`

  Set the client ID received by the `dc_clt_cltin_s` function.

- `trngid`

  Set the area that receives transaction global identifiers.

  Although a transaction global identifier consists of 16 characters, allocate an area of 17 bytes or greater, since NULL is appended to the end of the identifier.

- `trnbid`

  Set the area that receives transaction branch identifiers.

  Although a transaction branch identifier consists of 16 characters, allocate an area of 17 bytes or greater, since NULL is appended to the end of the identifier.

(4) Arguments whose values are returned

- `trngid`

  The transaction global identifier is returned.

- `trnbid`

  The transaction branch identifier is returned.

(5) Return values

Return Value	Value (decimal)	Meaning
`DC_OK`	0	Normal termination
`DCCLTER_INVALID_ARGS`	-2501	The pointer of `trngid` or `trnbid` is NULL.
`DCCLTER_PROTO`	-2502	The function has been issued from an invalid context (for example, from outside a transaction).
`DCCLTER_NO_BUFS`	-2504	Insufficient memory
`DCCLTER_INVALID_CLTID`	-2544	The client ID specified for `cltid` differs from the one received by the `dc_clt_cltin_s` function.

(6) Note

Specify an area of 17 bytes or greater for each of the `trngid` and `trnbid` arguments. If the area is smaller than 17 bytes, the area may be corrupted during TP1/Client internal processing.

4.5.7 dc_trn_info_s - Post information about current transaction

(1) Form

(a) TP1/Client/W

- **_s version of the function**

```
#include <dcvtrn.h>
DCLONG dc_trn_info_s(DCCLT_ID cltid, char *flags)
```

- **Non-_s version of the function**

```
#include <dcvtrn.h>
DCLONG dc_trn_info(char *flags)
```

(b) TP1/Client/P

- **_s version of the function**

```
#include <dcvtrn.h>
DCLONG dc_trn_info_s(DCCLT_ID cltid, char CLTFAR *flags)
```

- **Non-_s version of the function**

```
#include <dcvtrn.h>
DCLONG dc_trn_info(char CLTFAR *flags)
```

(2) Purpose

Posts whether the CUP having issued the `dc_trn_info_s` function remains active as a transaction.

(3) Arguments set by UAPs

- `cltid`

  Set the client ID received by the `dc_clt_cltin_s` function.

- `flags`

  Set NULL.

(4) Return values

Return Value	Value (decimal)	Meaning
-	1	The CUP process that issued the `dc_trn_info_s` function is in a transaction.
-	0	The CUP process that issued the `dc_trn_info_s` function is outside the transaction.
`DCCLTER_INVALID_CLTID`	-2544	The client ID specified in `cltid` differs from the one received by the `dc_clt_cltin_s` function.

Legend:

-: Not applicable

4.6 TCP/IP communication function

4.6.1 dc_clt_send_s - Sending messages

(1) Form

(a) TP1/Client/W

■ **_s version of the function**

```
#include <dcvclt.h>
DCLONG dc_clt_send_s(DCCLT_ID cltid, char *buff,
                     DCLONG sendleng, char *hostname,
                     unsigned short portnum, DCLONG flags)
```

■ **Non-_s version of the function**

```
#include <dcvclt.h>
int dc_clt_send(char *buff, DCLONG sendleng,
                char *hostname,
                unsigned short portnum, DCLONG flags)
```

(b) TP1/Client/P

■ **_s version of the function**

```
#include <dcvclt.h>
DCLONG dc_clt_send_s(DCCLT_ID cltid, char CLTFAR *buff,
                     DCLONG sendleng,
                     char CLTFAR *hostname,
                     unsigned short portnum, DCLONG flags)
```

■ **Non-_s version of the function**

```
#include <dcvclt.h>
int dc_clt_send(char CLTFAR *buff, DCLONG sendleng,
                char CLTFAR *hostname,
                unsigned short portnum, DCLONG flags)
```

(2) Purpose

Sends messages to the MHP.

Before issuing the `dc_clt_send_s` function, the `dc_rpc_open_s` function with flags set to `DCCLT_ONEWAY_SND` or `DCCLT_SNDRCV` must be issued.

(3) Arguments set by UAPs

- `cltid`

  Set the client ID received by the `dc_clt_cltin_s` function.

- `buff`

  Set the area that contains messages to send. The area must be greater than the length specified in `sendleng`.

- `sendleng`

  Set the length of a message to send.

- `hostname`

  Specify the host name of the node to be connected when no connection is established.

  If NULL is specified, the function accesses the contents of `DCSNDHOST` in the client environment definition that was acquired when the `dc_rpc_open_s` function was issued.

  You can specify a maximum of 63# characters for the host name.

  You can specify an IP address in decimal dot notation for the host name.

  # If you specify `00000008` for `DCCLTOPTION` in the client environment definition, you can specify a maximum of 255 characters for the host name.

- `portnum`

  Specify the port number of the node to be connected by establishing a connection when no connection is established.

  If `0` is specified, the function accesses the contents of `DCSNDPORT` in the client environment definition that was acquired when the `dc_rpc_open_s` function was issued.

- `flags`

  Specify whether to release the connection after sending message.

  `DCNOFLAGS`

  Does not release the connection after sending message.

  `DCCLT_SND_CLOSE`

  Releases the connection after sending message.

  Except error situations, specifying `DCNOFLAGS` maintains the connection until you issue the `dc_rpc_close_s` function.

(4) Return values

Return Value	Value (decimal)	Meaning
`DC_OK`	0	Normal termination
`DCCLTER_INVALID_ARGS`	-2501	Invalid argument
`DCCLTER_PROTO`	-2502	The `dc_rpc_open_s` function has not been issued. Or else, the `dc_rpc_open_s` function has been issued with `DCCLT_ONEWAY_SND` or `DCCLT_SNDRCV` not specified in flags.
`DCCLTER_NO_BUFS`	-2504	Insufficient memory
`DCCLTER_NET_DOWN`	-2506	Network error
`DCCLTER_TIMED_OUT`	-2507	A request for connection establishment timed out.
`DCCLTER_SYSERR`	-2518	System error
`DCCLTER_RESOURCE`	-2538	Insufficient resource
`DCCLTER_WRONG_HOST`	-2539	The host name is invalid, or has not been specified in both hostname and `DCSNDHOST`.
`DCCLTER_CONNREFUSED`	-2541	A connection establishment request to the remote system was rejected.
`DCCLTER_INVALID_CLTID`	-2544	The client ID specified for `cltid` differs from the one received by the `dc_clt_cltin_s` function.
`DCCLTER_PORT_IN_USE`	-2547	Port numbers that can be assigned automatically by the operating system are insufficient.

(5) Note

If the remote system releases the connection when the function sends a message to the remote system, depending on the length of the message, the function might not be able to detect that the connection has been released. However, a subsequent function might detect it. Keep this in mind when you create a CUP.

4.6.2 dc_clt_receive_s - Receiving messages

(1) Form

TP1/Client/W

(a) TP1/Client/W

■ _s version of the function

```
#include <dcvclt.h>
DCLONG dc_clt_receive_s(DCCLT_ID cltid, char *buff,
                        DCLONG recvleng, DCLONG timeout,
                        DCLONG flags)
```

■ Non-_s version of the function

```
#include <dcvclt.h>
int dc_clt_receive(char *buff, DCLONG recvleng,
                               DCLONG timeout,
                               DCLONG flags)
```

(b) TP1/Client/P

■ _s version of the function

```
#include <dcvclt.h>
DCLONG dc_clt_receive_s(DCCLT_ID cltid, char CLTFAR *buff,
                        DCLONG recvleng, DCLONG timeout,
                        DCLONG flags)
```

■ Non-_s version of the function

```
#include <dcvclt.h>
int dc_clt_receive(char CLTFAR *buff, DCLONG recvleng,
                   DCLONG timeout,
                   DCLONG flags)
```

(2) Purpose

Receives messages sent by an MHP.

Before issuing the`dc_clt_receive_s` function, the `dc_rpc_open_s` function with flags set to `DCCLT_ONEWAY_RCV` or `DCCLT_SNDRCV` must be issued.

(3) Arguments set by UAPs

- `cltid`

  Set the client ID received by the `dc_clt_cltin_s` function.

- `buff`

Set the area that contains messages received. The area must be greater than the length specified in `recvleng`.

- `recvleng`

  Set the length of a message to be received.

- `timeout`

  Set the maximum wait time (in seconds) for a message to receive. The value must be an integer from -1 to 65535.

  If `-1` is specified, the function waits indefinitely until a message comes. If `0` is specified, the function does not wait for a message. If there is no message to receive, the function returns with a `DCCLTER_TIMED_OUT` error.

  If any value between 1 and 65535 is specified, the function waits for a message to receive by the number of seconds specified. If no message can be received within the specified number of seconds, the function returns with a `DCCLTER_TIMED_OUT` error.

- `flags`

  Specify whether to release the connection after message reception.

  `DCNOFLAGS`

  Does not release the connection after message reception.

  `DCCLT_RCV_CLOSE`

  Releases the connection after message reception.

  Except error situations, specifying `DCNOFLAGS` maintains the connection until you issue the `dc_rpc_close_s` function.

(4) Argument returned

- `buff`

  Return the received message.

(5) Return values

Return Value	Value (decimal)	Meaning
`DC_OK`	0	Normal termination
`DCCLTER_INVALID_ARGS`	-2501	Invalid argument
`DCCLTER_PROTO`	-2502	The `dc_rpc_open_s` function has not been issued. Or else, the `dc_rpc_open_s` function has been issued with `DCCLT_ONEWAY_RCV` or `DCCLT_SNDRCV` not specified in flags.

Return Value	Value (decimal)	Meaning
`DCCLTER_NO_BUFS`	-2504	Insufficient memory
`DCCLTER_NET_DOWN`	-2506	Network error
`DCCLTER_TIMED_OUT`	-2507	Timeout occurred during reception of the message.
`DCCLTER_SYSERR`	-2518	System error
`DCCLTER_RESOURCE`	-2538	Insufficient resource
`DCCLTER_CONNFREE`	-2542	Connection was freed by the remote system.
`DCCLTER_INVALID_CLTID`	-2544	The client ID specified for `cltid` differs from the one received by the `dc_clt_cltin_s` function.

(6) Notes

- The `dc_clt_receive_s` function returns control to the CUP if any of the following events occurs:
  - A message of the length specified in `recvleng` is received from the MHP.

    Control is not returned to the CUP if the received message is shorter than the specified length.
  - A timeout error occurs when a message from the MHP is received.
  - The MHP frees the connection.

    (Control is not returned to the CUP if the received message is shorter than the specified length.)
  - A network error occurs.
- If the MHP frees the connection when the `dc_clt_receive_s` function is issued, the function returns with a `DCCLTER_CONNFREE` error.

4.6.3 dc_clt_receive2_s - Receiving messages (messages receivable even if an error occurs)

(1) Form

(a) TP1/Client/W

■ _s version of the function

```
#include <dcvclt.h>
DCLONG dc_clt_receive2_s(DCCLT_ID cltid, char *buff,
                         DCLONG *recvleng, DCLONG timeout,
                         DCLONG flags)
```

■ Non-_s version of the function

```
#include <dcvclt.h>
DCLONG dc_clt_receive2(char *buff, DCLONG *recvleng,
                       DCLONG timeout, DCLONG flags)
```

(b) TP1/Client/P

■ _s version of the function

```
#include <dcvclt.h>
DCLONG dc_clt_receive2_s(DCCLT_ID cltid, char CLTFAR *buff,
                         DCLONG CLTFAR *recvleng,
                         DCLONG timeout,
                         DCLONG flags)
```

■ Non-_s version of the function

```
#include <dcvclt.h>
DCLONG dc_clt_receive2(char CLTFAR *buff,
                       DCLONG CLTFAR *recvleng,
                       DCLONG timeout, DCLONG flags)
```

(2) Purpose

Receives messages from the MHP.

Before issuing the `dc_clt_receive2_s` function, you need to issue the `dc_rpc_open_s` function with flags set to `DCCLT_ONEWAY_RCV` or `DCCLT_SNDRCV`.

(3) Arguments set by UAPs

- `cltid`

  Set the client ID received by the `dc_clt_cltin_s` function.

- `buff`

  Set the area that contains messages to receive. The area must be greater than the length specified in `recvleng`.

- `recvleng`

  Set the length of a message to receive.

- `timeout`

Set the maximum wait time in seconds for receiving messages. The timeout value must be an integer between -1 and 65535.

Specifying `-1` lets the program wait indefinitely until it receives a message. Specifying `0` disables the program from waiting for messages. When no messages are available for reception, the function returns an error with `DCCLTER_TIMED_OUT`.

Specifying `1` to `65535` allows the program to wait for messages for the specified seconds. When no messages are received within the time, the function returns an error with `DCCLTER_TIMED_OUT`.

- `flags`

  Specify whether to release the connection after message reception.

  `DCNOFLAGS`

  Does not release the connection after message reception.

  `DCCLT_RCV_CLOSE`

  Releases the connection after message reception.

  Except error situations, specifying `DCNOFLAGS` maintains the connection until you issue the `dc_rpc_close_s` function.

(4) Argument returned from

- `buff`

  Return the received message.

- `recvleng`

  Return the length of the received message.

(5) Return values

Return Value	Value (decimal)	Meaning
`DC_OK`	0	Normal termination
`DCCLTER_INVALID_ARGS`	-2501	Invalid argument
`DCCLTER_PROTO`	-2502	The `dc_rpc_open_s` function has not been issued. Alternatively the `dc_rpc_open` function has been issued with `DCCLT_ONEWAY_RCV` or `DCCLT_SNDRCV` not specified in flags.
`DCCLTER_NO_BUFS`	-2504	Insufficient memory
`DCCLTER_NET_DOWN`	-2506	Network error

Return Value	Value (decimal)	Meaning
DCCLTER_TIMED_OUT	-2507	Timeout occurred during reception of the message.
DCCLTER_SYSERR	-2518	System error
DCCLTER_RESOURCE	-2538	Insufficient resource
DCCLTER_CONNFREE	-2542	The remote system released the connection.
DCCLTER_INVALID_CLTID	-2544	The client ID specified for cltid differs from the one received by the dc_clt_cltin_s function.

(6) Notes

- The dc_clt_receive2_s function returns control to the CUP when:
  - The program has received a message for the full length specified in recvleng from the MHP.

    (Control is not returned to the CUP if the received message is shorter than the specified length.)
  - A timeout error occurs during message reception from the MHP.
  - The MHP releases the connection.
  - A network error occurs.
- When the dc_clt_receive2_s function is issued, a disconnection from the MHP allows the function to return an error with DCCLTER_CONNFREE.

4.6.4 dc_clt_assem_send_s - Sending assembled messages

(1) Form

(a) _s version of the function

```
#include <dcvclt.h>
DCLONG dc_clt_assem_send_s(DCCLT_ID cltid, char CLTFAR *buff,
DCLONG sendleng,
char CLTFAR *hostname, unsigned short portnum, DCLONG timeout,
DCLONG flags)
```

(b) Non-_s version of the function

```
#include <dcvclt.h>
DCLONG dc_clt_assem_send(char CLTFAR *buff, DCLONG sendleng,
char CLTFAR *hostname,
unsigned short portnum, DCLONG timeout, DCLONG flags)
```

(2) Purpose

The `dc_clt_assem_send_s` function uses the message assembly facility to send messages. When this facility is used, the function sends four-byte message information followed by the message body specified in the `buff` argument. If a connection to the remote system has not been established, the function first establishes the connection according to the values specified in the `hostname` and `portnum` arguments.

If `Y` is specified for `DCCLTDELIVERYCHECK` of the client environment definition, the function also uses the message delivery confirmation facility. In this case, the size of the message information sent before the message body is 11 bytes. After receiving the 11-byte message information, TP1/Client returns control to the CUP.

Before issuing this function, make sure that you issue the `dc_rpc_open_s` function in which `DCCLT_ONEWAY_SND` or `DCCLT_SNDRCV` is specified for the `flags` argument.

(3) Arguments set by UAPs

- `cltid`

  Specify the client ID received by the `dc_clt_cltin_s` function.

- `buff`

  Specify the area that contains the message to be sent. The area must be larger than the length specified in `sendleng`.

- `sendleng`

  Specify the length of the message to be sent.

- `hostname`

  Specify the host name of the node to be connected if no connection has been established.

  If `NULL` is specified, the function accesses the contents of `DCSNDHOST` in the client environment definition acquired when the `dc_rpc_open_s` function was issued.

  You can specify a maximum of 63# characters for the host name.

  You can also specify an IP address in decimal dot notation for the host name.

  #:

  > If you specify `00000008` for `DCCLTOPTION` in the client environment definition, you can specify a maximum of 255 characters for the host name.

- `portnum`

Specify the port number of the node to be connected when there is no connection and a connection must be established.

If `0` is specified, the function accesses the contents of `DCSNDPORT` in the client environment definition acquired when the `dc_rpc_open_s` function was issued.

- `timeout`

  This argument takes effect when the message delivery confirmation facility is used. Specify the maximum time (in seconds) that the function waits for response-only data to arrive. The value must be an integer from -1 to 65,535.

  If `-1` is specified:

  The function waits indefinitely for response-only data.

  If `0` is specified:

  The function does not wait for response-only data. If there is no message to be received, the function returns a `DCCLTER_TIMED_OUT` error.

  If any value from 1 to 65,535 is specified:

  The function waits for a message, but returns a `DCCLTER_TIMED_OUT` error if a message does not arrive within the specified number of seconds.

  If divided response-only data arrives, the function repeats the receive processing until 11-byte response-only data arrives. The timeout specified by this argument is applied every time the function attempts reception. If you want to use the value of this argument as the maximum response wait time for the client, specify the `00000002` option for `DCCLTOPTION` of the client environment definition.

- `flags`

  Specify whether to release the connection after sending a message.

  `DCNOFLAGS`

  After a message is sent, the connection is not released until the `dc_rpc_close_s` function is issued (exception: an error occurs).

  `DCCLT_RCV_CLOSE`

  After a message is sent, the connection is released. If the message delivery confirmation facility is being used, the connection is released after message information has been received.

(4) Return values

Return value	Value (decimal)	Meaning
`DC_OK`	0	Normal termination

Return value	Value (decimal)	Meaning
DCCLTER_INVALID_ARGS	-2501	Invalid argument
DCCLTER_PROTO	-2502	Possible causes are as follows: • The dc_rpc_open_s function has not been issued. • The dc_rpc_open_s function was issued but neither DCCLT_ONEWAY_SND nor DCCLT_SNDRCV was specified for the flags argument.
DCCLTER_NO_BUFS	-2504	Insufficient memory
DCCLTER_NET_DOWN	-2506	A network error occurred. The connection is released.
DCCLTER_TIMED_OUT	-2507	A connection establishment request timed out. Alternatively, reception of response-only data timed out when the message delivery confirmation facility was being used. The connection is released.
DCCLTER_SYSERR	-2518	A system error occurred. If the error is a network error, the connection is released.
DCCLTER_RESOURCE	-2538	Insufficient resource
DCCLTER_WRONG_HOST	-2539	The host name is incorrect. Alternatively, a host name is not specified in either hostname or DCSNDHOST.
DCCLTER_CONNREFUSED	-2541	A connection establishment request to the remote system was rejected.
DCCLTER_CONNFREE	-2542	The connection was released by the remote system when the message delivery confirmation facility was being used.
DCCLTER_INVALID_CLTID	-2544	The client ID specified for cltid differs from the one received by the dc_clt_cltin_s function.
DCCLTER_PORT_IN_USE	-2547	Port numbers that can be assigned automatically by the operating system are insufficient.
DCCLTER_INVALID_MESSAGE	-2548	An invalid message was received when the message delivery confirmation facility was being used. The connection is released.
DCCLTER_COLLISION_MESSAGE	-2584	Messages collided when the message delivery confirmation facility was being used. The connection is released.

(5) Notes

- If the remote system releases the connection when the function sends a message to the remote system, depending on the length of the message, the function might not be able to detect that the connection has been released. The following describes what occurs in this case according to the facility used:

  When the message assembly facility is used:

  > If the function fails to detect the release of a connection when it sends a message, a subsequent function might detect the release. This must be kept in mind when a CUP is created.

  When the message delivery confirmation facility is used:

  > If the function fails to detect the release of a connection when it sends a message, the function detects the release when it receives response-only data.

- If the message assembly and message delivery confirmation facilities are used, short packets are used for sending and receiving. As a result, transmission processing might take more time. If more time might be required, specify `Y` for `DCCLTTCPNODELAY` of the client environment definition.

4.6.5 dc_clt_assem_receive_s - Receiving assembled messages

(1) Form

(a) _s version of the function

```
#include <dcvclt.h>
DCLONG dc_clt_assem_receive_s(DCCLT_ID cltid, char CLTFAR
*buff, DCLONG CLTFAR *recvleng,
DCLONG timeout, DCLONG flags)
```

(b) Non-_s version of the function

```
#include <dcvclt.h>
DCLONG dc_clt_assem_receive(char CLTFAR *buff, DCLONG CLTFAR
*recvleng,
DCLONG timeout, DCLONG flags)
```

(2) Purpose

The `dc_clt_assem_receive_s` function uses the message assembly facility to receive messages. When this facility is used, the function receives four-byte message information, and then receives data equivalent to the size set in the message information. The function then stores the data in the `buff` argument. The four-byte message information is not stored in the `buff` argument. The length of the received message is stored in the `recvleng` argument. The message length stored in the `recvleng` argument does not include the message information length.

If `Y` is specified for `DCCLTDELIVERYCHECK` of the client environment definition, the message delivery confirmation facility is used when a message is sent or received. In this case, the function receives 11-byte message information, and then receives data equivalent to the size set in the message information. The function then stores the data in the `buff` argument. The 11-byte message information is not stored in the `buff` argument. The length of the received message is stored in the `recvleng` argument. The message length stored in the `recvleng` argument does not include the message information length. If the received message information includes a response request, the function sends 11-byte message information, and then returns control to the CUP.

Before issuing this function, make sure that you issue the `dc_rpc_open_s` function in which `DCCLT_ONEWAY_RCV` or `DCCLT_SNDRCV` is specified for the `flags` argument.

(3) Arguments set by UAPs

- `cltid`

  Specify the client ID received by the `dc_clt_cltin_s` function.

- `buff`

  Specify the area where the received message will be stored. The area must be larger than the length of the message sent by the remote system.

- `recvleng`

  Specify the length of the area where the received message will be stored (the area specified by `buff`).

- `timeout`

  Specify the maximum time (in seconds) that the function waits for a message to arrive. The value must be an integer from -1 to 65,535.

  If `-1` is specified:

    The function waits indefinitely for a message.

  If `0` is specified:

    The function does not wait for a message. If there is no message to be received, the function returns a `DCCLTER_TIMED_OUT` error.

  If any value from 1 to 65,535 is specified:

    The function waits for a message, but returns a `DCCLTER_TIMED_OUT` error if no message arrives within the specified number of seconds.

If a divided message arrives, the function repeats the receive processing until the entire message arrives. The timeout specified by this argument is applied every time the function attempts a reception. If you want to use the value of this argument as the maximum response wait time for the client, specify the `00000002` option for `DCCLTOPTION` of the client environment definition.

- `flags`

  Specify whether to release the connection after receiving a message.

  `DCNOFLAGS`

  After a message is received, the connection is not released until the `dc_rpc_close_s` function is issued (exception: an error occurs).

  `DCCLT_RCV_CLOSE`

  After a message is received, the connection is released. If the message delivery confirmation facility is being used, the connection is released after message information has been sent.

(4) Arguments that contain return values

- `buff`

  The received message is stored in the area specified by this argument. The stored message does not include message information. If a timeout occurs, the data received before the timeout is stored.

- `recvleng`

  The length of the received message is stored in the area specified by this argument. The stored length does not include the length of the message information. If a timeout occurs, the length of the data received before the timeout is stored.

(5) Return values

Return value	Value (decimal)	Meaning
`DC_OK`	0	Normal termination
`DCCLTER_INVALID_ARGS`	-2501	Invalid argument
`DCCLTER_PROTO`	-2502	Possible causes are as follows: • The `dc_rpc_open_s` function has not been issued. • The `dc_rpc_open_s` function was issued but neither `DCCLT_ONEWAY_RCV` nor `DCCLT_SNDRCV` was specified for the `flags` argument.
`DCCLTER_NO_BUFS`	-2504	Insufficient memory

Return value	Value (decimal)	Meaning
`DCCLTER_NET_DOWN`	-2506	A network error occurred. The connection is released.
`DCCLTER_TIMED_OUT`	-2507	Message reception timed out. The connection is released.
`DCCLTER_SYSERR`	-2518	A system error occurred. If the error is a network error, the connection is released.
`DCCLTER_RESOURCE`	-2538	Insufficient resource
`DCCLTER_CONNFREE`	-2542	The connection was released by the remote system.
`DCCLTER_INVALID_CLTID`	-2544	The client ID specified for `cltid` differs from the one received by the `dc_clt_cltin_s` function.
`DCCLTER_INF_TOO_BIG`	-2546	The area prepared by the CUP was too small to receive the message from the remote system. The connection is released.
`DCCLTER_INVALID_MESSAGE`	-2548	An invalid message was received. The connection is released.

(6) Notes

- This function returns control to the CUP in the following cases only:
    - When the function has received message data equivalent to the length set in the message information
    - When a network error has occurred
    - When message reception has timed out
    - When the connection is released by the remote system
    - When the message storage area (specified by the `buff` argument) is too small to hold the message sent by the remote system
    - When an invalid message has been received
- If the message assembly and message delivery confirmation facilities are used, short packets are used for sending and receiving. As a result, transmission processing might take more time. If more time might be required, specify `Y` for `DCCLTTCPNODELAY` of the client environment definition.

4.7 Facility for receiving one-way messages from the server

4.7.1 dc_clt_accept_notification_s - One-way message reception

(1) Form

(a) TP1/Client/W

■ **_s version of the function**

```
#include <dcvclt.h>
DCLONG dc_clt_accept_notification_s(
                           HWND hWnd, char *defpath,
                           char *inf,
                           DCLONG *inf_len,
                           unsigned short port,
                           DCLONG timeout,
                           char *hostname,
                           char *nodeid, DCLONG flags)
```

■ **Non-_s version of the function**

```
#include <dcvclt.h>
DCLONG dc_clt_accept_notification(
                           char *inf, DCLONG *inf_len,
                           unsigned short port,
                           DCLONG timeout,
                           char *hostname, char *nodeid,
                           DCLONG flags)
```

(b) TP1/Client/P

■ **_s version of the function**

```
#include <dcvclt.h>
DCLONG dc_clt_accept_notification_s(
                           HWND hWnd, char CLTFAR *defpath,
                           char CLTFAR *inf,
                           DCLONG CLTFAR *inf_len,
                           unsigned short port,
                           DCLONG timeout,
                           char CLTFAR *hostname,
                           char CLTFAR *nodeid,
                           DCLONG flags)
```

■ Non-_s version of the function

```
#include <dcvclt.h>
DCLONG dc_clt_accept_notification(
                           char CLTFAR *inf,
                           DCLONG CLTFAR *inf_len,
                           unsigned short port,
                           DCLONG timeout,
                           char CLTFAR *hostname,
                           char CLTFAR *nodeid,
                           DCLONG flags)
```

(2) Purpose

This function waits for the message reported by the `dc_rpc_cltsend` function executed on the server side. This function stops waiting for the message if a timeout occurs before receiving the message. The timeout is specified by the `timeout` argument. On reception of the message, this function returns the return value, received message, host name of the message-originating server, and node identifier of the message-originating server, and control returns to the CUP. Before issuing this function, you do not need to issue the `dc_clt_cltin_s` and `dc_rpc_open_s` functions.

(3) Arguments set by UAPs

■ `hWnd`

Specify NULL.

■ `defpath`

Specify the path name of the client environment definition file. The path name must be specified with the full path or with a relative path from the current drive and the current directory. The following shows the order in which files are loaded when the path name is specified.

- In TP1/Client/P

  Client environment definition files are loaded in the following order:

  1. The `BETRAN.INI` file in the Windows directory

  2. The client environment definition file specified in the `defpath` argument

  The definitions in both the client environment definition file and the `BETRAN.INI` file take effect.

  If the same definition is specified in each file with a different value, the value specified in the client environment definition file takes effect.

  If neither the client environment definition file nor the `BETRAN.INI` file contains the necessary specification, TP1/Client/P uses the defaults.

- In TP1/Client/W

  All definitions specified in the environment variables will be invalid. TP1/Client/W uses the defaults for definitions that are not specified in the client environment definition file specified in the `defpath` argument.

You can omit the path name by specifying NULL at the beginning of the `defpath` argument. The following describes the operation when the path name is omitted.

- In TP1/Client/P

  TP1/Client/P uses the `BETRAN.INI` file in the Windows directory as the client environment definition file. If the `BETRAN.INI` file does not exist or if the contents of the definition file are invalid, TP1/Client/P uses the defaults.

- In TP1/Client/W

  TP1/Client/W uses the values specified in the environment variables. If an environment variable is not specified, TP1/Client/W uses the default.

The following describes operation when the client environment definition file specified in the `defpath` argument does not exist or when the contents of the definition file are invalid.

- In TP1/Client/P

  TP1/Client/P uses the `BETRAN.INI` file in the Windows directory as the client environment definition file. If the `BETRAN.INI` file does not exist or if the contents of the definition file are invalid, TP1/Client/P uses the defaults.

- In TP1/Client/W

  TP1/Client/W uses the defaults. The values specified in the environment variables will be invalid.

■ `inf`

Specify the area for storing the message sent from the server.

■ `inf_len`

Specify the length of the area for storing the message sent from the server. That is, specify the length of the `inf` argument. You can specify a value in the range from 0 to `DCRPC_MAX_MESSAGE_SIZE`#.

# If you specify 2 or a larger value for `DCCLTRPCMAXMSGSIZE` in the client environment definition, the value you specify is used rather than the value of `DCRPC_MAX_MESSAGE_SIZE` (1 megabyte).

- `port`

  Specify a client's port number between 5001 and 65535. Specify a unique port number for each process or thread if multiple processes or multiple threads are executed simultaneously on the same machine.

- `timeout`

  Specify a timeout value in seconds between 0 and 65535. Value 0 means an infinite wait.

- `hostname`

  Specify an area of 64 bytes# or more for storing the host name of the server that sent the message. The host name is not stored if you specify NULL.

  # If you specify `00000008` for `DCCLTOPTION` in the client environment definition, this value is 256 bytes, not 64 bytes.

- `nodeid`

  Specify the 8-byte area for storing the node identifier of the server that sent the message.

- `flags`

  Specify `DCNOFLAGS`.

(4) Arguments returned

- `inf`

  Notification message from the server.

- `inf_len`

  Notification message length from the server.

- `hostname`

  Host computer name for the server that notified the message. An IP address in decimal dot notation is returned if resolution to a host name has failed. This value is not returned if you specify NULL.

- `nodeid`

  Node identifier for the server that notified the message. The node identifier is suffixed by a NULL character as shown below.

Node identifier (4 bytes)	NULL character (4 bytes)

(5) Return values

Return Value	Value (decimal)	Meaning
`DC_OK`	0	Normal termination
`DCCLTER_INVALID_ARGS`	-2501	Invalid argument
`DCCLTER_FATAL`	-2503	Unsuccessful initialization. Alternatively, the client environment definition is specified incorrectly.
`DCCLTER_NO_BUFS`	-2504	A necessary amount of buffer could not be allocated.
`DCCLTER_NET_DOWN`	-2506	Network error
`DCCLTER_TIMED_OUT`	-2507	A timeout occurred during reception of the message.
`DCCLTER_SYSERR`	-2518	System error
`DCCLTER_VERSION`	-2535	Different versions
`DCCLTER_INF_TOO_BIG`	-2546	The received message is too large for the CUP-provided area. The part that does not fit is truncated. Values have already been set in the `hostname` and `nodeid` arguments.
`DCCLTER_PORT_IN_USE`	-2547	The specified port number is already used.
`DCCLTER_INVALID_MESSAGE`	-2548	Invalid message received
`DCCLTER_ACCEPT_CANCELED`	-2549	The one-way message reception wait status was canceled by the `dc_clt_cancel_notification_s` function. Values have already been set in the `inf`, `inf_len`, and `hostname` arguments.

(6) Notes

- Specify an area of 64 bytes# or more for the `hostname` argument, and an area of 8 bytes or greater for the `nodeid` argument. If the area is smaller than the required value, the area may be corrupted during TP1/Client internal processing.

  # If you specify `00000008` for `DCCLTOPTION` in the client environment definition, this value is 256 bytes, not 64 bytes.

- Specify a unique port number in the `port` argument for each process or thread when multiple processes or multiple threads are executed simultaneously on the same machine. Do not specify a port number for use by the operating system or other programs even if one can be specified in the `port` argument. If you specify a port number in this case, response data might not be received correctly. The port numbers used by the operating system depend on the operating system. For details, see the documentation of your operating system.

- In TP1/Client, you can use a different client environment definition for each `dc_clt_accept_notification_s` function call. To do so, create a separate client environment definition file for each `dc_clt_accept_notification_s` function call, and specify the file name in the `defpath` argument of the function.

4.7.2 dc_clt_cancel_notification_s - Canceling one-way message wait

(1) Form

(a) TP1/Client/W

■ _s version of the function

```
#include <dcvclt.h>
DCLONG dc_clt_cancel_notification_s(
                           HWND hWnd, char *defpath,
                           char *inf, DCLONG inf_len,
                           unsigned short port,
                           char *hostname, DCLONG flags)
```

■ Non-_s version of the function

```
#include <dcvclt.h>
DCLONG dc_clt_cancel_notification(
                          char *inf, DCLONG inf_len,
                          unsigned short port, char
                          *hostname,
                          DCLONG flags)
```

(b) TP1/Client/P

■ _s version of the function

```
#include <dcvclt.h>
DCLONG dc_clt_cancel_notification_s(
                         HWND hWnd, char CLTFAR *defpath,
                         char CLTFAR *inf, DCLONG inf_len,
                         unsigned short port,
                         char CLTFAR *hostname, DCLONG
                         flags)
```

■ Non-_s version of the function

```
#include <dcvclt.h>
DCLONG dc_clt_cancel_notification(
char CLTFAR *inf, DCLONG inf_len,
unsigned short port, char CLTFAR
*hostname,
DCLONG flags)
```

(2) Purpose

Cancels the wait for receiving a one-way message from the server. The wait state is enabled by the `dc_clt_accept_notification_s` function.

When canceling the wait state, you can send the message specified for `inf` to the CUP that waits for a one-way message.

(3) Arguments set by UAPs

■ `hWnd`

Specify NULL.

■ `defpath`

Specify the path name of the client environment definition file. The path name must be specified with the full path or a path from the current drive and the current directory. The following shows the order in which files are loaded when the path name is specified.

- In TP1/Client/P

  Client environment definition files are loaded in the following order:

  1. The `BETRAN.INI` file in the Windows directory

  2. The client environment definition file specified in the `defpath` argument

  The definitions in both the client environment definition file and the `BETRAN.INI` file take effect.

  If the same definition is specified in each file with a different value, the value specified in the client environment definition file takes effect.

  If neither the client environment definition file nor the `BETRAN.INI` file contains the necessary specification, TP1/Client/P uses the defaults.

- In TP1/Client/W

  All definitions specified in the environment variables will be invalid. TP1/Client/W uses the defaults for definitions that are not specified in the client environment definition file specified in the `defpath` argument.

You can omit the path name by specifying NULL at the beginning of the `defpath` argument. The following describes the operation when the path name is omitted.

- In TP1/Client/P

TP1/Client/P uses the BETRAN.INI file in the Windows directory as the client environment definition file. If the BETRAN.INI file does not exist or if the contents of the definition file are invalid, TP1/Client/P uses the defaults.

- In TP1/Client/W

  TP1/Client/W uses the values specified in the environment variables. If an environment variable is not specified, TP1/Client/W uses the default.

The following describes operation when the client environment definition file specified in the defpath argument does not exist or when the contents of the definition file are invalid.

- In TP1/Client/P

  TP1/Client/P uses the BETRAN.INI file in the Windows directory as the client environment definition file. If the BETRAN.INI file does not exist or if the contents of the definition file are invalid, TP1/Client/P uses the defaults.

- In TP1/Client/W

  TP1/Client/W uses the defaults. The values specified in the environment variables will be invalid.

■ inf

Specify a message notified to the CUP.

■ inf_len

Specify the message length (inf length). Available values range from 0 to DCRPC_MAX_MESSAGE_SIZE#. Specifying 0 notifies no messages to the CUP.

# If you specify 2 or a larger value for DCCLTRPCMAXMSGSIZE in the client environment definition, the value you specify is used rather than the value of DCRPC_MAX_MESSAGE_SIZE (1 megabyte).

■ port

Specify a port number for the one-way message request between 5001 and 65535.

■ hostname

Specify the name of the host on which the CUP is waiting for one-way messages. You can specify a maximum of 63# characters for the host name.

You can specify an IP address in decimal dot notation for the host name.

# If you specify 00000008 for DCCLTOPTION in the client environment definition, you can specify a maximum of 255 characters for the host name.

■ `flags`

Specify `DCNOFLAGS`.

(4) Return values

Return Value	Value (decimal)	Meaning
`DC_OK`	0	Normal termination
`DCCLTER_INVALID_ARGS`	-2501	Invalid argument
`DCCLTER_FATAL`	-2503	Initialization failed. Alternatively, the client environment definition is specified incorrectly.
`DCCLTER_NO_BUFS`	-2504	A necessary amount of buffer could not be allocated. Alternatively, resources became insufficient.
`DCCLTER_NET_DOWN`	-2506	Network error
`DCCLTER_SERVICE_NOT_UP`	-2514	The CUP is not in the one-way message reception wait status.
`DCCLTER_SYSERR`	-2518	System error
`DCCLTER_WRONG_HOST`	-2539	Invalid host computer name
`DCCLTER_PORT_IN_USE`	-2547	The specified port number is in use, or port numbers that can be assigned automatically by the operating system are insufficient.

(5) Note

In TP1/Client, you can use a different client environment definition for each `dc_clt_cancel_notification_s` function call. To do so, create a separate client environment definition file for each `dc_clt_cancel_notification_s` function call, and specify the file name in the `defpath` argument of the function.

4.7.3 dc_clt_open_notification_s - Start reception of one-way messages

(1) Form

(a) TP1/Client/W

■ **_s version of the function**

```
#include <dcvclt.h>
DCLONG dc_clt_open_notification_s(HWND hWnd,
                                  DCCLT_ID *ntfid,
                                  char *defpath,
                                  unsigned short port,
                                       DCLONG flags)
```

■ Non-_s version of the function

```
#include <dcvclt.h>
DCLONG dc_clt_open_notification(unsigned short port,
                                DCLONG flags)
```

(b) TP1/Client/P

■ _s version of the function

```
#include <dcvclt.h>
DCLONG dc_clt_open_notification_s(HWND hWnd,
                                  DCCLT_ID CLTFAR *ntfid,
                                  char CLTFAR *defpath,
                                  unsigned short port,
                                  DCLONG flags)
```

■ Non-_s version of the function

```
#include <dcvclt.h>
DCLONG dc_clt_open_notification(unsigned short port,
                                DCLONG flags)
```

(2) Purpose

The `dc_clt_open_notification_s` function creates an environment for using the facility for receiving one-way messages from the server.

The `dc_clt_open_notification_s` and `dc_clt_close_notification_s` functions are used in a pair.

(3) Arguments set by UAPs

- `hWnd`

  Specify NULL.

- `ntfid`

  Specify a pointer to the area for receiving the one-way message reception ID.

- `defpath`

  Specify the path name of the client environment definition file. The path name must be specified with the full path or with a relative path from the current drive and the current directory. The following shows the order in which files are loaded when the path name is specified.

- In TP1/Client/P

  Client environment definition files are loaded in the following order:

  1. The `BETRAN.INI` file in the Windows directory

  2. The client environment definition file specified in the `defpath` argument

  The definitions in both the client environment definition file and the `BETRAN.INI` file take effect.

  If the same definition is specified in each file with a different value, the value specified in the client environment definition file takes effect.

  If neither the client environment definition file nor the `BETRAN.INI` file contains the necessary specification, TP1/Client/P uses the defaults.

- In TP1/Client/W

  All definitions specified in the environment variables will be invalid. TP1/Client/W uses the defaults for definitions that are not specified in the client environment definition file specified in the `defpath` argument.

You can omit the path name by specifying NULL at the beginning of the `defpath` argument. The following describes the operation when the path name is omitted.

- In TP1/Client/P

  TP1/Client/P uses the `BETRAN.INI` file in the Windows directory as the client environment definition file. If the `BETRAN.INI` file does not exist or if the contents of the definition file are invalid, TP1/Client/P uses the defaults.

- In TP1/Client/W

  TP1/Client/W uses the values specified in the environment variables. If an environment variable is not specified, TP1/Client/W uses the default.

The following describes operation when the client environment definition file specified in the `defpath` argument does not exist or when the contents of the definition file are invalid.

- In TP1/Client/P

  TP1/Client/P uses the `BETRAN.INI` file in the Windows directory as the client environment definition file. If the `BETRAN.INI` file does not exist or if the contents of the definition file are invalid, TP1/Client/P uses the defaults.

- In TP1/Client/W

TP1/Client/W uses the defaults. The values specified in the environment variables will be invalid.

- `port`

  Specify a client's port number between 5001 and 65535. Specify a unique port number for each process or thread when multiple processes or multiple threads are executed simultaneously on the same machine.

- `flags`

  Specify `DCNOFLAGS`.

(4) Arguments specifying the containers of returned values

- `ntfid`

  Specifies the area for containing the returned one-way message reception ID.

(5) Return values

Return value	Value (decimal)	Meaning
`DC_OK`	0	The function normally terminated.
`DCCLTER_INVALID_ARGS`	-2501	The value specified in an argument is incorrect.
`DCCLTER_PROTO`	-2502	The `dc_clt_open_notification` function has already been executed. This value is not returned if the `dc_clt_open_notification_s` function is executed.
`DCCLTER_FATAL`	-2503	Initialization failed. Alternatively, the client environment definition is specified incorrectly.
`DCCLTER_NO_BUFS`	-2504	A necessary amount of buffer could not be allocated.
`DCCLTER_PORT_IN_USE`	-2547	The specified port number has already been used.

(6) Notes

- After the `dc_clt_open_notification_s` function is terminated normally, always issue the `dc_clt_close_notification_s` function. If the `dc_clt_close_notification_s` function is not issued, the resource used by the `dc_clt_open_notification_s` function may remain.
- Specify a unique port number in the `port` argument for each process or thread when multiple processes or multiple threads are executed simultaneously on the same machine. Do not specify a port number for use by the operating system or other programs even if one can be specified in the `port` argument. If you specify a port number in this case, response data might not be received correctly. The port numbers used by the operating system depend on the operating system. For

details, see the documentation of your operating system.

- In TP1/Client, you can use a different client environment definition for each `dc_clt_open_notification_s` function call. To do so, create a separate client environment definition file for each `dc_clt_open_notification_s` function call, and specify the file name in the `defpath` argument of the function.

4.7.4 dc_clt_close_notification_s - Terminate reception of one-way messages

(1) Form

(a) _s version of the function

```
#include <dcvclt.h>
DCLONG dc_clt_close_notification_s(DCCLT_ID ntfid, DCLONG
flags)
```

(b) Non-_s version of the function

```
#include <dcvclt.h>
DCLONG dc_clt_close_notification(DCLONG flags)
```

(2) Purpose

The `dc_clt_close_notification_s` function deletes the environment for using the facility for receiving one-way messages from the server.

The `dc_clt_open_notification_s` and `dc_clt_close_notification_s` functions are used in a pair.

(3) Arguments set by UAPs

- `ntfid`

  Specify the one-way message reception ID received by the `dc_clt_open_notification_s` function.

- `flags`

  Specify `DCNOFLAGS`.

(4) Return values

Return value	Value (decimal)	Meaning
`DC_OK`	0	The function normally terminated.
`DCCLTER_INVALID_ARGS`	-2501	The value specified in an argument is incorrect.

Return value	Value (decimal)	Meaning
`DCCLTER_NO_BUFS`	-2504	A necessary amount of buffer could not be allocated.
`DCCLTER_INVALID_NTFID`	-2544	The one-way message reception ID specified in `ntfid` differs from that received by the `dc_clt_open_notification_s` function.

4.7.5 dc_clt_chained_accept_notification_s - Receive a one-way message

(1) Form

(a) TP1/Client/W

■ _s version of the function

```
#include <dcvclt.h>
DCLONG dc_clt_chained_accept_notification_s
                             (DCCLT_ID ntfid, char *inf,
                              DCLONG *inf_len,
                              DCLONG timeout,
                              char *hostname,
                              char *nodeid,
                              DCLONG flags)
```

■ Non-_s version of the function

```
#include <dcvclt.h>
DCLONG dc_clt_chained_accept_notification
                       (char *inf, DCLONG *inf_len,
                        DCLONG timeout, char *hostname,
                        char *nodeid, DCLONG flags)
```

(b) TP1/Client/P

■ _s version of the function

```
#include <dcvclt.h>
DCLONG dc_clt_chained_accept_notification_s
                             (DCCLT_ID ntfid, char CLTFAR
                              *inf,
                              DCLONG CLTFAR *inf_len,
                              DCLONG timeout,
                              char CLTFAR *hostname,
                              char CLTFAR *nodeid,
                              DCLONG flags)
```

■ **Non-_s version of the function**

```
#include <dcvclt.h>
DCLONG dc_clt_chained_accept_notification
                       (char CLTFAR *inf, DCLONG CLTFAR
                        *inf_len,
                        DCLONG timeout, char CLTFAR
                        *hostname,
                        char CLTFAR *nodeid, DCLONG flags)
```

(2) Purpose

This function waits for the message reported by the `dc_rpc_cltsend` function executed on the server side. This function stops waiting for the message if a timeout occurs before receiving the message. The timeout is specified by the `timeout` argument. On reception of the message, this function returns the return value, received message, host name of the message-originating server, and node identifier of the message-originating server, and control returns to the CUP.

Before issuing the `dc_clt_chained_accept_notification_s` function, always issue the `dc_clt_open_notification_s` function.

(3) Arguments set by UAPs

■ `ntfid`

Specify the one-way message reception ID received by the `dc_clt_open_notification_s` function.

■ `inf`

Specify the area for storing the message sent from the server.

■ `inf_len`

Specify the length of the area for storing the message sent from the server. That is, specify the length of the `inf` argument. You can specify a value in the range from 0 to `DCRPC_MAX_MESSAGE_SIZE`#.

# If you specify 2 or a larger value for `DCCLTRPCMAXMSGSIZE` in the client environment definition, the value you specify is used rather than the value of `DCRPC_MAX_MESSAGE_SIZE` (1 megabyte).

■ `timeout`

Specify the timeout in seconds. You can specify 0 to 65535. When 0 is specified, a timeout does not occur.

- `hostname`

  Specify an area of 64 bytes# or more for storing the host name of the server that sent the message. The host name is not stored if you specify NULL.

  # If you specify `00000008` for `DCCLTOPTION` in the client environment definition, this value is 256 bytes, not 64 bytes.

- `nodeid`

  Specify the 8-byte area for storing the node identifier of the server that sent the message.

- `flags`

  Specify `DCNOFLAGS`.

(4) Arguments that contain return values

- `inf`

  The notification message from the server is returned.

- `inf_len`

  The length of the notification message from the server is returned.

- `hostname`

  The host name of the message-originating server is returned.

  If translation to a host name fails, the IP address is returned in the dotted decimal format. This value is not returned if you specify NULL.

- `nodeid`

  The node identifier of the message-originating server is returned. The node identifier is suffixed by a NULL character as shown below.

Node identifier (4 bytes)	NULL character (4 bytes)

(5) Return values

Return value	Value (decimal)	Meaning
`DC_OK`	0	The function normally terminated.
`DCCLTER_INVALID_ARGS`	-2501	The value specified in an argument is incorrect.
`DCCLTER_PROTO`	-2502	The `dc_clt_open_notification_s` function has not been executed.
`DCCLTER_NO_BUFS`	-2504	A necessary amount of buffer could not be allocated.

Return value	Value (decimal)	Meaning
`DCCLTER_NET_DOWN`	-2506	A network error occurred.
`DCCLTER_TIMED_OUT`	-2507	A timeout occurred before a message arrived.
`DCCLTER_SYSERR`	-2518	A system error occurred.
`DCCLTER_VERSION`	-2535	Versions do not match.
`DCCLTER_INVALID_NTFID`	-2544	The one-way message reception ID specified in `ntfid` differs from that received by the `dc_clt_open_notification_s` function.
`DCCLTER_INF_TOO_BIG`	-2546	The received message is too large for the CUP-provided area. The part that does not fit is truncated. Values have already been set in the `hostname` and `nodeid` arguments.
`DCCLTER_INVALID_MESSAGE`	-2548	An invalid message was received.
`DCCLTER_ACCEPT_CANCELED`	-2549	The one-way message reception wait status was canceled by the `dc_clt_cancel_notification_s` function. Values have already been set in the `inf`, `inf_len`, and `hostname` arguments.

(6) Notes

Specify an area of 64 bytes# or more for the `hostname` argument, and an area of 8 bytes or greater for the `nodeid` argument. If the area is smaller than the required value, the area may be corrupted during TP1/Client internal processing.

# If you specify `00000008` for `DCCLTOPTION` in the client environment definition, this value is 256 bytes, not 64 bytes.

4.8 XATMI interface facility

4.8.1 tpalloc - Allocate typed buffer

(1) Form

(a) TP1/Client/W

```
#include <dcvxatmi.h>
char *tpalloc(char *type, char *subtype, DCLONG size)
```

(b) TP1/Client/P

```
#include <dcvxatmi.h>
char CLTFAR *tpalloc(char CLTFAR *type,
      char CLTFAR *subtype, DCLONG size)
```

(2) Purpose

Allocates the typed buffer.

Some types of buffer must be initialized before use. The `tpalloc` function initializes the buffer after the buffer is allocated until it returns. The buffer is returned to the issuer of the `tpalloc` function when issuance of a function becomes available.

Define the method of initialization by communication resource managers for TP1/Client and TP1/Server Base. If not defined, the `tpalloc` function does not initialize the buffer.

When the initialization completes successfully, the `tpalloc` function returns a pointer to a buffer of the appropriate type aligned on a long word. The function returns NULL if an error occurs; error information is returned as the return value. If initialization failed, the allocated buffer is released and NULL is returned.

(3) Arguments set by UAPs

- `type`

  Specify `X_OCTET` as the buffer type.

- `subtype`

  Specify NULL as the buffer subtype.

- `size`

  Specify the size of the buffer to be allocated.

(4) Return values

When the initialization completes successfully, the `tpalloc` function returns a pointer to a buffer of the appropriate type aligned on a long word. If an error occurs, the function returns NULL and sets one of the following values in `tperrno` as a return value to report the information about the error.

Return value	Meaning
`TPEINVAL`	Invalid argument
`TPENOENT`	The value specified in the argument is not defined in the system.
`TPEPROTO`	Inappropriate status for issuing the `tpalloc` function.
`TPESYSTEM`	An error occurred in the communication resource manager.
`TPEOS`	An error occurred in the operating system.

(5) Notes

- The `tpalloc` function cannot be used with any of these C functions: `malloc`, `realloc`, or free.

  Example:

  The buffer allocated by the `tpalloc` function cannot be released by the free function.

  Operation of the system is not guaranteed when above functions are used together.
- The buffer returned by the `tpalloc` function is initialized to zero.
- The buffer area is acquired from the global heap.
- When `TPESYSTEM` is returned for an error in TP1/Client, error information is output to the error log.
- When `TPEOS` is returned, insufficient memory is suspected as the cause. Error information is output to the error log when the error occurred in TP1/Client.

4.8.2 tpfree - Release typed buffer

(1) Form

(a) TP1/Client/W

```
#include <dcvxatmi.h>
void tpfree(char *ptr)
```

(b) TP1/Client/P

```
#include <dcvxatmi.h>
void tpfree(char CLTFAR *ptr)
```

(2) Purpose

Releases the typed buffer allocated by `tpalloc`.

The `tpfree` function does not return the return value to the caller. The function must be specified in the void type.

The typed buffer is not released when NULL is specified in the argument `ptr`. Processing results of the `tpfree` function are not guaranteed when the value specified in `ptr` is not the pointer to the typed buffer or has already been released by the `tpfree` function.

For the buffer type used for requesting information or when it is linked to data, releasing the buffer also deletes its additional information. The `tpfree` function deletes the linkage of additional information before releasing the buffer. Define the method of deletion of additional information by communication resource managers for TP1/Client and TP1/Server Base.

Once the `tpfree` function returns, the argument specified in `ptr` can no longer be passed to an XATMI interface function as a new argument or be referenced.

(3) Argument set by UAPs

■ `ptr`

Specify the pointer to the typed buffer allocated by the `tpalloc` function.

(4) Notes

- The `tpfree` function cannot be used with any of these C functions: `malloc`, `realloc`, or free.

  Example:

  The buffer allocated by the `malloc` function cannot be released by the `tpfree` function.

  Operation of the system is not guaranteed when above functions are used together.

4.8.3 tpconnect - Establish connection with interactive service

(1) Form

(a) TP1/Client/W

```
#include <dcvxatmi.h>
DCLONG tpconnect(char *svc, char *data, DCLONG len,
                 DCLONG flags)
```

(b) TP1/Client/P

```
#include <dcvxatmi.h>
DCLONG tpconnect(char CLTFAR *svc, char CLTFAR *data,
                 DCLONG len, DCLONG flags)
```

(2) Purpose

Establishes half-duplex connection between TP1/Client and an interactive service. When the function is processed normally, the descriptor that specifies the connection is returned.

The issuer of the `tpconnect` function can pass the specified information to the service function to the receiver during establishment of connection. To pass information, the pointer to the buffer allocated by the `tpalloc` function must be specified in `data`, and the data length to send must be specified in `len`.

The `tpconnect` function allows information to be received under the interactive service without issuing the function for receiving data.

(3) Arguments set by UAPs

- `svc`

  Specify the service name of the service to request.

- `data`

  Specify `X_OCTET` as the pointer to the typed buffer that contains send data.

- `len`

  Specify the length of data to be sent. The maximum length is 500 x 1024 bytes. Set `0` when the length need not be specified. Do not set `0` for the buffer whose length must be specified.

- `flags`

  Specify any of the following.

  `TPNOTRAN`

  When the issuer of the function is in the transaction mode, the started service does not belong to the issuer's transaction.

  Be sure to specify `TPNOTRAN` when the issuer of the function in the transaction mode requests the service that belongs to the server unavailable for transaction processing.

  When the issuer of the function is in the transaction mode, a transaction timeout error may occur even if `TPNOTRAN` is specified.

Failure in the service started with `TPNOTRAN` will not affect the issuer's transaction.

`TPSENDONLY`

Connection is established first so that the issuer can send data and the service called by the function can only receive data. The called service first gains control of connection.

`TPRECVONLY`

Connection is established first so that the issuer can receive data and the service called by the function can only send data. The called service first gains control of connection.

Either `TPSENDONLY` or `TPRECVONLY` must be specified.

`TPNOBLOCK`

When the blocking status occurs (e.g., the internal buffer is filled with messages sent), neither connection is established nor data is sent.

If the blocking status occurs with `TPNOBLOCK` not specified, the issuer remains blocked until the cause of blocking is removed or a transaction or blocking timeout error occurs.

`TPNOTIME`

The issuer is infinitely blocked to prevent blocking timeout error.

Transaction timeout error may occur even if `TPNOTIME` is specified.

`TPSIGRSTRT`

System call interrupted by a signal during execution is recalled.

(4) Return values

When the processing completes successfully, the `tpconnect` function returns a descriptor to specify the established connection. If an error occurs, the function returns `-1` and sets one of the following values in `tperrno` as a return value to report the information about the error.

Return value	Meaning
`TPEINVAL`	Invalid argument
`TPENOENT`	Since the value specified in the argument is not defined in the system, connection cannot be established.
`TPEITYPE`	The value specified in the argument cannot be used in the specified service.
`TPELIMIT`	Since the number of unsolved connections reached to the limit, the request from the caller is not sent.

Return value	Meaning
`TPETRAN`	The specified service belongs to the server unavailable for transaction processing, but `TPNOTRAN` is not specified.
`TPETIME`	A timeout error occurred. • For the issuer in the transaction mode: A transaction timeout error occurred. The process terminates abnormally. The transaction is rolled back. `TPETIME` is returned to any message sent or received by any connection until rollback is completed. • For the issuer in other than the transaction mode: A blocking timeout error occurred where neither `TPNOBLOCK` nor `TPNOTIME` is specified.
`TPEBLOCK`	Blocking status occurred when the `tpconnect` function was issued with `TPNOBLOCK` specified.
`TPGOTSIG`	The signal is received, but `TPSIGRSTRT` is not specified.
`TPEPROTO`	Inappropriate status for issuing the `tpconnect` function.
`TPESYSTEM`	An error occurred in the communication resource manager.
`TPEOS`	An error occurred in the operating system.

(5) Notes

- When communication is disabled due to the blocking status under OpenTP1, `TPESYSTEM` is returned as well as for communication disabled due to network failure.
- When information unavailable to the service is specified under OpenTP1, `TPESYSTEM` is returned. When the issuer of the function is in the transaction mode, the transaction is rolled back.
- Unless otherwise specified for X/Open, the error that needs rollback of the transaction under OpenTP1 is `TPESYSTEM`. Some transactions may not be rolled back if `TPESYSTEM` is returned.
- When the service request is not authenticated under the OpenTP1 security facility, `TPEPROTO` is returned. Check the detailed error code of the UAP trace to find the cause of the error.
- TP1/Client cannot issue the `tpconnect` function in a transaction unless permanent connection has been established with the facility of establishing permanent connection.
- If a transaction timeout error occurred under TP1/Client, the CUP execution process terminates abnormally and all the connections established before the timeout error are disconnected. `TPETIME` is returned only for blocking timeout error.

- When `TPESYSTEM` is returned for an error in TP1/Client, error information is output to the error log.
- When `TPEOS` is returned, insufficient memory is suspected. Error information is output to the error log when the error occurred in TP1/Client.

4.8.4 tpdiscon - Disconnect connection with interactive service

(1) Form

TP1/Client/W or TP1/Client/P

```
#include <dcvxatmi.h>
int tpdiscon (DCLONG cd)
```

(2) Purpose

Disconnects connection with interactive service, and reports the event (`TPEV_DISCONIMM`) to the interactive service.

Issuing the `tpdiscon` function immediately disconnects connection. Data that does not reach the destination is discarded. The `tpdiscon` function can also be used when the interactive service belongs to the transaction of the issuer. In this case, the transaction is rolled back.

The `tpdiscon` function can be issued from only the originator of the interactive service. This function cannot be issued within the interactive service. The issuer of this function does not need to have a right to control the connection.

The TP1/Client interactive service issues the `tpdiscon` function that reports the disconnection when either system completes communication.

(3) Argument set by UAPs

■ `cd`

Specify the descriptor of the interactive service for which disconnection is to be reported.

(4) Return values

If an error occurs, the `tpdiscon` function returns `-1` and sets one of the following values in `tperrno` as a return value to report the information about the error.

Return value	Meaning
`TPEBADDESC`	The argument is invalid, or is specified as the descriptor of the called interactive service.
`TPETIME`	Timeout error occurred. The specified descriptor is invalidated.
`TPEPROTO`	Inappropriate status for issuing the `tpdiscon` function.

Return value	Meaning
TPESYSTEM	An error occurred in the communication resource manager.
TPEOS	An error occurred in the operating system.

(5) Notes

- TPETIME is not returned under OpenTP1.
- When TPESYSTEM is returned for an error in TP1/Client, error information is output to the error log.
- When TPEOS is returned, insufficient memory is suspected. Error information is output to the error log when the error occurred in TP1/Client.

4.8.5 tpsend - Send message to interactive service

(1) Form

(a) TP1/Client/W

```
#include <dcvxatmi.h>
int tpsend(DCLONG cd, char *data, DCLONG len,
           DCLONG flags,
           DCLONG *revent)
```

(b) TP1/Client/P

```
#include <dcvxatmi.h>
int tpsend(DCLONG cd, char CLTFAR *data, DCLONG len,
    DCLONG flags, DCLONG CLTFAR *revent)
```

(2) Purpose

Sends data to the interactive service.

The system with control of connection can issue the tpsend function.

(3) Arguments set by UAPs

- cd

  Specify the connection for sending data. Specify the descriptor indicated by the return value of the tpconnect function.

  When an event is set for cd, the tpsend function terminates without sending data, assuming that processing failed.

- data

Specify `X_OCTET` as the pointer to the typed buffer that contains data to be sent.

Specifying NULL results in error.

Specify the same value that defined in the interactive service.

- `len`

  Specify the length of data to be sent. The maximum length is 500 x 1024 bytes. Specify `0` for the address of the pointer to the buffer for which the length need not be specified. Do not specify `0` for the buffer whose length must be specified.

- `flags`

  Specify any of the following.

  `TPRECVONLY`

  The issuer of the `tpsend` function aborts control of connection after sending data. The issuer can no longer issue the `tpsend` function. When the receiver at the other end of connection receives data sent by the `tpsend` function, it also receives the event (`TPEV_SENDONLY`) that indicates control of connection. The data receiver can no longer issue the `tprecv` function.

  `TPNOBLOCK`

  When the blocking status occurred (e.g., the internal buffer is filled with messages sent), neither data nor event is sent.

  If the blocking status occurs with `TPNOBLOCK` not specified, the issuer of the `tpsend` function remains blocked until the communication is resumed or a transaction or blocking timeout error occurs.

  `TPNOTIME`

  The issuer of the `tpsend` function is blocked infinitely. Blocking timeout error will never occur. However, transaction timeout error may occur.

  `TPSIGRSTRT`

  System call interrupted by a signal during execution is recalled.

- `revent`

  Specify the pointer to the typed buffer that indicates an event.

  The following events can be returned by the `tpsend` function.

  `TPEV_DISCONIMM`

  The connection is immediately disconnected by the `tpdiscon` function issued by the initiator. Or, it is disconnected due to a communication error such as failure of the server, machine, or network.

  When the connection is disconnected by the `tpdiscon` function, TPEV_DISCONIMM is reported to the remote system. When the connection is disconnected due to a communication error, TPEV_DISCONIMM is returned to both the initiator and the remote system.

TPEV_SVCERR

The remote system that has no control of connection issued the tpreturn function. TPEV_SVCERR is returned to the initiator.

TPEV_SVCFAIL

The remote system that has no control of connection issued the tpreturn function. TPEV_SVCFAIL is returned to the initiator.

Also, the tpreturn function was called without TPFAIL or data specified. TPFAIL is specified in rval and NULL in data.

These events indicate that the connection was disconnected immediately, causing all data to be lost. The descriptor used for the connection is invalidated. The transaction that contains the two programs is rolled back.

(4) Return values

If an error occurs, the tpsend function returns -1 and sets one of the following values in tperrno as a return value to report the information about the error.

Return value	Meaning
TPEINVAL	Invalid argument
TPEBADDESC	Invalid descriptor is specified in cd.
TPETIME	A timeout error occurred. • For the issuer in the transaction mode: A transaction timeout error occurred. The transaction is rolled back. In this case, TPETIME is returned to new data transmission or undetermined response until the transaction is rolled back. • For the issuer in other than the transaction mode: A blocking timeout error occurred where neither TPNOBLOCK nor TPNOTIME is specified. In either case, the value specified in *data is not changed.
TPEEVENT	An event occurred. The return value is returned to revent.
TPEBLOCK	Blocking status occurred when the tprecv function was issued with TPNOBLOCK specified.
TPGOTSIG	The signal is received, but TPSIGRSTRT is not specified.
TPEPROTO	Inappropriate status for issuing the tpsend function.
TPESYSTEM	An error occurred in the communication resource manager.

Return value	Meaning
TPEOS	An error occurred in the operating system.

(5) Notes

- TPNOBLOCK is invalid for OpenTP1. TPEBLOCK is not returned. When communication is disabled due to the blocking status under OpenTP1, TPESYSTEM is returned as well as for communication disabled due to shutdown of network.
- TPNOTIME is invalid for OpenTP1.
- TPSIGRSTRT is invalid. Whether or not TPSIGRSTRT is specified does not affect operation. When a signal is received, processing is interrupted and the system call is recalled. TPGOTSIG is not returned.
- If a transaction timeout error occurred under OpenTP1, the process terminates abnormally. TPETIME is returned only for a blocking timeout error.
- Unless otherwise specified for X/Open, the error that needs rollback of the transaction under OpenTP1 is TPESYSTEM. Some transactions may not be rolled back if TPESYSTEM is returned.
- OpenTP1 cannot report an event by tpsend function even if the remote system of the service issued the tpdiscon or tpreturn function, unless the event has been received by the process that calls the tpsend function.
- If a transaction timeout error occurred under TP1/Client, the CUP execution process terminates abnormally and all the connections established before the timeout error are disconnected. TPETIME is returned only for a blocking timeout error.
- When TPESYSTEM is returned for an error in TP1/Client, error information is output to the error log.
- When TPEOS is returned, insufficient memory is suspected to be the cause. Error information is output to the error log when the error occurred in TP1/Client.

4.8.6 tprecv - Receive message from interactive service

(1) Form

(a) TP1/Client/W

```
#include <dcvxatmi.h>
int tprecv(DCLONG cd, char *CLTFAR *data,
           DCLONG CLTFAR *len,
           DCLONG flags, DCLONG CLTFAR *revent)
```

(b) TP1/Client/P

```
#include <dcvxatmi.h>
int tprecv(DCLONG cd, char CLTFAR *CLTFAR *data,
      DCLONG CLTFAR *len, DCLONG flags,
      DCLONG CLTFAR *revent)
```

(2) Purpose

Receives data from the interactive service.

The system without control of connection can issue the `tprecv` function.

When the `tprecv` function returns with `TPEV_SVCSUCC` or `TPEV_SVCFAIL` specified in `revent`, the value passed by the application as the `tpreturn` function argument can be referenced as the global variable `tpurcode`.

(3) Arguments set by UAPs

- `cd`

  Specify the connection for receiving data. Specify the descriptor indicated by the return value of the `tpconnect` function.

- `data`

  Specify `X_OCTET` as the pointer to the typed buffer that contains received data.

  Specifying NULL results in error.

- `len`

  Specify the length of data to be received. The maximum length is 500 x 1024 bytes. If the specified value is greater than the total buffer length before the `tprecv` function is issued, a new value is set in `len`. If no data received, `0` is set.

- `flags`

  Specify any of the following.

  `TPNOCHANGE`

  The buffer type specified in data is not converted.

  The buffer type of received data must match that specified in data. When `TPNOCHANGE` is not specified, the value of data is converted to the buffer type of the received data.

  `TPNOBLOCK`

  The `tprecv` function does not wait until data arrives.

  The `tprecv` function receives data and returns if data is ready for reception.

When TPNOBLOCK is not specified and data is not ready for reception, the issuer of the function is blocked until data arrives.

TPNOTIME

The issuer of the function is blocked infinitely. Blocking timeout error will never occur. However, transaction timeout error may occur.

TPSIGRSTRT

System call interrupted by a signal during execution is recalled.

- revent

Specify the pointer to the typed buffer that stores events.

When an event is set for cd, the type of event is returned to revent. The value specified in data can be received with the TPEV_SVCSUCC, TPEV_SVCFAIL, and TPEV_SENDONLY events.

The following events can be specified for the tprecv function.

TPEV_DISCONIMM

The connection is immediately disconnected by the tpdiscon function issued by the initiator of the interactive service. Or, it is disconnected due to a communication error.

When the connection is disconnected by the tpdiscon function, TPEV_DISCONIMM is returned to the remote system. When the connection is disconnected due to a communication error in which the server, machine, and network failed, TPEV_DISCONIMM is returned to both the initiator and the remote system.

TPEV_DISCONIMM is returned immediately after the disconnection, and data being sent is aborted. The transaction is rolled back. In this case, the descriptor used for the connection is invalid.

TPEV_SENDONLY

The system at the other end of connection has aborted control of connection. The system that receives TPEV_SENDONLY can send but cannot receive data until it aborts control of connection.

TPEV_SVCERR

The remote system of the service initiator has issued the tpreturn function. Any of the following error occurred during execution of the tpreturn function.

An invalid argument is passes to the tpreturn function.

The tpreturn function is issued while the service was opening connection.

When TPEV_SVCERR is returned, data or return values defined for the application will not affect the operation. The connection is disconnected and the value of cd is invalidated. If TPEV_SVCERR occurred in the transaction of the receiver of data, the transaction is rolled back.

TPEV_SVCFAIL

The service of the remote system was specified by the application, but terminated without being completed (the tpreturn function returns TPFAIL). TPEV_SVCFAIL is returned to the initiator.

If the remote system's service has control of connection when the tpreturn function is issued, the service can pass the typed buffer to the initiator.

The server disconnects connection when the tprecv function terminates; the value of cd is invalidated. If TPEV_SVCFAIL occurred in the transaction of the receiver of data, the transaction is rolled back.

TPEV_SVCSUCC

The service of the remote system at the other end of service was specified by the application, and terminated after being completed (the tpreturn function returns TPSUCCESS). TPEV_SVCSUCC is returned to the initiator.

The server disconnects connection when the tprecv function terminates; the value of cd is invalidated. If TPEV_SVCSUCC occurred in the transaction of the receiver of data, the transaction is committed or rolled back by the server, depending on the transaction mode.

(4) Return values

If an error occurs, the tprecv function returns -1 and sets one of the following values in tperrno as a return value to report the information about the error.

Return value	Meaning
TPEINVAL	Invalid argument
TPEBADDESC	Invalid descriptor is specified in cd.
TPEOTYPE	The issuer of the tprecv function does not identify the buffer type arrived. Alternatively, the buffer type specified in data does not match the sent buffer type when TPNOCHANGE is specified in the flags argument. In either case, the value of data or len remains unchanged. When the service is executed as a transaction of the issuer of the tprecv function, the transaction is rolled back until the arrived buffer is aborted. If the above error occurred, the specified event is aborted and the processing result of the service is undetermined. The issuer must terminate the service immediately.

Return value	Meaning
TPETIME	A timeout error occurred. • For the issuer in the transaction mode: A transaction timeout error occurred. The transaction is rolled back. In this case, TPETIME is returned to new data transmission or undetermined response until the transaction is rolled back. • For the issuer in other than the transaction mode: A blocking timeout error occurred where neither TPNOBLOCK nor TPNOTIME is specified. In either case, the value specified in data is not changed.
TPEEVENT	An event occurred. The return value is returned to revent.
TPEBLOCK	Blocking status occurred when the tprecv function was issued with TPNOBLOCK specified.
TPGOTSIG	The signal is received, but TPSIGRSTRT is not specified.
TPEPROTO	Inappropriate status for issuing the tpdiscon function.
TPESYSTEM	An error occurred in the communication resource manager.
TPEOS	An error occurred in the operating system.

(5) Notes

- When a signal is received, processing is interrupted and the system call is recalled. TPGOTSIG is not returned. Whether or not TPSIGRSTRT is specified does not affect the calling of system call.
- If a transaction timeout error occurred under OpenTP1, the process terminates abnormally. TPETIME is returned only for a blocking timeout error.
- Unless otherwise specified for X/Open, the error that needs rollback of the transaction under OpenTP1 is TPESYSTEM. Some transactions may not be rolled back if TPESYSTEM is returned.
- If a transaction timeout error occurred under TP1/Client, the CUP execution process terminates abnormally and all the connections established before the timeout error are disconnected. TPETIME is returned only for a blocking timeout error.
- The tpconnect function cannot be issued to TP1/Client. Only the descriptor returned by the tpconnect function can be specified in cd.
- When TPESYSTEM is returned for an error in TP1/Client, error information is output to the error log.
- When TPEOS is returned, insufficient memory is suspected to be the cause. Error information is output to the error log when the error occurred in TP1/Client.

4.9 Character code converter (When not using a code mapping table)

The character code converter provides only the non-_s version of functions. However, these functions operate normally even in a multi-thread environment.

The character code converter is only available for TP1/Client/P.

4.9.1 dc_clt_code_convert - Converting character codes

(1) Form

TP1/Client/W DLL or TP1/Client/P DLL

```
#include <dcvclt.h>
DCLONG dc_clt_code_convert(DCLONG request,
      char CLTFAR *source, DCULONG CLTFAR *source_len,
      char CLTFAR *dest, DCULONG CLTFAR *dest_len,
      DCLONG flags)
```

(2) Purpose

- Converts the character strings consisting of JIS code or Shift JIS code into character strings of EBCDIC code, EBCDIK code, or KEIS code.
- Converts the character strings consisting of EBCDIC code, EBCDIK code, or KEIS code into character strings of JIS code or Shift JIS code.

(3) Arguments set by UAPs

■ `request`

Specify the conversion method using the following request code.

`DCCLT_JISSJIS_TO_EBCKEIS`

Converts character strings consisting of JIS code or Shift JIS code into character strings of EBCDIC code, EBCDIK code, or KEIS code.

`DCCLT_EBCKEIS_TO_JISSJIS`

Converts character strings consisting of EBCDIC code, EBCDIK code, or KEIS code into character strings of JIS code or Shift JIS code.

■ `source`

Specify the character string to be converted.

■ `source_len`

Specify the length of the character string to be converted. 1 to `DCRPC_MAX_MESSAGE_SIZE` can be specified.

- `dest`

  Specify the area that receives data after conversion.

- `dest_len`

  Specify the length of the area that receives the converted character string. 1 to `DCRPC_MAX_MESSAGE_SIZE` can be specified.

- `flags`

  Specify the conditions for conversion using the following format (OR of the specified values).

  When 2., 3., 4., 5. or 6. comes first, omit | (stroke).

  {`DCNOFLAGS`   }

  [1.][|2.][|3.][|4.][|5.][|6.]

  1:{`DCCLT_CNV_EBCDIC` | <u>`DCCLT_CNV_EBCDIK`</u>}

  2:{`DCCLT_CNV_SPCHAN` | <u>`DCCLT_CNV_SPCZEN`</u>}

  3:{`DCCLT_CNV_KEIS78` | <u>`DCCLT_CNV_KEIS83`</u>}

  4:{`DCCLT_CNV_INVSPC` | <u>`DCCLT_CNV_INVERR`</u>}

  5:{`DCCLT_CNV_TAB` | <u>`DCCLT_CNV_NOTAB`</u>}

  6:{`DCCLT_CNV_CNTL` | <u>`DCCLT_CNV_NOCNTL`</u>}

  Description of the specified values

  `DCNOFLAGS`

  > The following defaults are used.
  >
  > EBCDIK code is used.
  >
  > Two-byte spaces remain the same.
  >
  > The 1983 version of the KEIS code is used.
  >
  > An error occurs if an invalid code is found.
  >
  > A tab code is not identified to be single-byte. No shift code is available even for just the preceding or succeeding two-byte code if any.
  >
  > A control code is not identified to be single-byte. No shift code is available even for just the preceding or succeeding two-byte code if any.

  `DCCLT_CNV_EBCDIC`

EBCDIC code is used.

`DCCLT_CNV_EBCDIK`

EBCDIK code is used.

`DCCLT_CNV_KEIS78`

The 1978 version of the KEIS code is used.

`DCCLT_CNV_KEIS83`

The 1983 version of the KEIS code is used.

`DCCLT_CNV_INVSPC`

An invalid code is converted to a space.

`DCCLT_CNV_INVERR`

An error occurs if an invalid code is found.

`DCCLT_CNV_TAB`

Identifies a tab code to be single-byte. A shift code is given to just the preceding or succeeding two-byte code if any.

`DCCLT_CNV_NOTAB`

Does not identify a tab code to be single-byte. No shift code is provided for just the preceding or succeeding two-byte code if any.

`DCCLT_CNV_CNTL`

Identifies a control code to be single-byte. A shift code is given to just the preceding or succeeding two-byte code if any.

`DCCLT_CNV_NOCNTL`

Does not identify a control code to be single-byte. No shift code is provided for just the preceding or succeeding two-byte code if any.

(4) Arguments for which a value is returned

- `dest`

  The converted character string is returned.

- `dest_len`

  The length of the converted character string is returned.

(5) Return values

Return Value	Value (decimal)	Meaning
`DC_OK`	0	Normal termination
`DCCLTER_INVALID_ARGS`	-2501	Invalid argument
`DCCLTER_NO_BUFS`	-2504	Insufficient memory. The function returns this value also when the specified character length covers the first byte of a two-byte code that is contained in the character string to be converted.
`DCCLTER_INVALID_CODE`	-2550	An invalid code is found in the character string.
`DCCLTER_OVERFLOW`	-2551	The length of the converted character string exceeds the area prepared by the CUP.

(6) Note

When you specify request to be `DCCLT_EBCKEIS_TO_JISSJIS` and flags to be `DCCLT_CNV_TAB` or `DCCLT_CNV_CNTL`, you need to prepare data that contains single-byte tab and control codes.

4.10 Character code converter (When using a code mapping table)

The character code converter provides only the non-_s version of functions. However, these functions operate normally even in a multi-thread environment.

The character code converter is only available for TP1/Client/P.

4.10.1 dc_clt_codeconv_open - Starting character code conversion

(1) Form

TP1/Client/W DLL or TP1/Client/P DLL

```
#include <dcvclt.h>
DCLONG dc_clt_codeconv_open(char CLTFAR *defpath,
      DCULONG CLTFAR *cnthd1,DCLONG flags)
```

(2) Purpose

Starts character code conversion to allocate a code mapping table to be used in the memory.

(3) Arguments set by UAPs

- `defpath`

  Specify NULL.

- `cnthdl`

  Specify a pointer to an area for receiving the handle of a control table to be used for character code conversion.

- `flags`

  Specify the conversion method.

  `DCNOFLAGS`

  Performs conversion by operations without using a code mapping table.

  `DCCLT_CNV_CommuniNet`

  Links with CommuniNet for conversion.

(4) Argument for which a value is returned

- `cnthdl`

  This argument returns the handle of a character code conversion control table allocated in the memory.

(5) Return values

Return Value	Value (decimal)	Meaning
`DC_OK`	0	Normal termination
`DCCLTER_INVALID_ARGS`	-2501	The value set for the argument is invalid.
`DCCLTER_NO_BUFS`	-2504	Insufficient memory size
`DCCLTER_NOFILE`	-2557	A code mapping table is not found.
`DCCLTER_NOT_SUPPORTED`	-2558	This value means that using the code mapping table is not supported. This value is also returned when the code mapping table has never been saved using the CommuniNet code mapping utility after the installation of CommuniNet.
`DCCLTER_FILE_IO`	-2559	An I/O error occurred in the code mapping table.

(6) Notes

- The use of this function requires a CommuniNet code mapping table. Before using this function, create a code mapping table using the CommuniNet code mapping utility.
- You cannot use a code mapping table using the CommuniNet code mapping utility unless you first save the table after the installation of CommuniNet. Before using this function, save a code mapping table using the CommuniNet code mapping utility.
- The filename of a CommuniNet code mapping table must be `CMAPEX.TBL`. Store the code mapping table under a Windows directory before using this function.
- The processing of the character code converter does not reflect the changes in the contents of a code mapping table made by the CommuniNet code mapping utility during the use of this function.
- This function does not save error logs and UAP trace information.
- Issue the function for starting character code conversion (`dc_clt_codeconv_open()`) only once for code conversion (`dc_clt_codeconv_exec()`). Do not issue the function for starting character code conversion more than once to prevent memory shortage. If you issue two or more functions, issue one function for terminating character code conversion (`dc_ctl_codeconv_close()`) for each of the issued functions.

4.10.2 dc_clt_codeconv_close - Terminating character code conversion

(1) Form

TP1/Client/W DLL or TP1/Client/P DLL

```
#include <dcvclt.h>
DCLONG dc_clt_codeconv_close(DCULONG cnthd1, DCLONG flags)
```

(2) Purpose

Terminates character code conversion to release the area on the memory that contains a code mapping table allocated.

(3) Arguments set by UAPs

- `cnthdl`

  Specify the handle of the control table acquired by the `dc_clt_codeconv_open` function for character code conversion.

- `flags`

  Specify `DCNOFLAGS`.

(4) Return values

Return Value	Value (decimal)	Meaning
`DC_OK`	0	Normal termination
`DCCLTER_INVALID_ARGS`	-2501	A value set for the argument is invalid.
`DCCLTER_NO_BUFS`	-2504	Insufficient memory

(5) Notes

- The use of this function requires a CommuniNet code mapping table. Before using this function, create a code mapping table using the CommuniNet code mapping utility.
- You cannot use a code mapping table unless you first save the table using the CommuniNet code mapping utility after the installation of CommuniNet. Before using this function, save the code mapping table using the CommuniNet code mapping utility.
- The filename of a CommuniNet code mapping table must be `CMAPEX.TBL`. Store the code mapping table under a Windows directory before using this function.

- The processing by the character code converter does not reflect changes made in a code mapping table by the CommuniNet code mapping utility during the use of this function.
- This function does not save error logs and UAP trace information.
- Issue the function for starting character code conversion (`dc_clt_codeconv_open()`) only once for code conversion (`dc_clt_codeconv_exec()`). Do not issue the function for starting character code conversion more than once to prevent memory shortage. If you issue two or more functions, issue one function for terminating character code conversion (`dc_ctl_codeconv_close()`) for each of the issued functions.

4.10.3 dc_clt_codeconv_exec - Executing character code conversion

(1) Form

TP1/Client/W DLL or TP1/Client/P DLL

```
#include <dcvclt.h>
DCLONG dc_clt_codeconv_exec(DCLONG request,
                           char CLTFAR *source,
                           DCULONG CLTFAR *source_len,
                           char CLTFAR *dest,
                           DCULONG CLTFAR *dest_len,
                           DCULONG cnthdl, DCLONG flags)
```

(2) Purpose

Executes character code conversion as follows:

Converts character strings consisting of JIS code or Shift JIS code into character strings of EBCDIC, EBCDIK or KEIS codes. Converts character strings consisting of EBCDIC, EBCDIK or KEIS code into character strings of JIS or Shift JIS code.

(3) Arguments set by UAPs

■ `request`

Specify the conversion method using the following request code.

`DCCLT_JISSJIS_TO_EBCKEIS`

Converts character strings consisting of JIS or shift JIS codes into character strings of EBCDIC, EBCDIK or KEIS code.

`DCCLT_EBCKEIS_TO_JISSJIS`

Converts character strings consisting of EBCDIC, EBCDIK or KEIS code into character strings of JIS or Shift JIS code.

- `source`

  Specify the character string to be converted.

- `source_len`

  Specify the length of the character string to be converted. You can specify a value from `1` to `DCRPC_MAX_MESSAGE_SIZE`.

- `dest`

  Specify the area that receives data after conversion.

- `dest_len`

  Specify the length of the area that receives the converted character string. You can specify a value from `1` to `DCRPC_MAX_MESSAGE_SIZE`.

- `cnthdl`

  Specify the handle of the control table acquired by `dc_clt_codeconv_open()` for code conversion.

- `flags`

  Specify the conditions for conversion in the following format (logical addition of the specified values). Omit a stroke (`|`) 2., 3., 4., 5. or 6. that comes first.

  {`DCNOFLAGS`   }

  `[`1.`][|`2.`][|`3.`][|`4.`][|`5.`][|`6.`]`

  1:{`DCCLT_CNV_EBCDIC` | <u>`DCCLT_CNV_EBCDIK`</u>}

  2:{`DCCLT_CNV_SPCHAN` | <u>`DCCLT_CNV_SPCZEN`</u>}

  3:{`DCCLT_CNV_KEIS78` | <u>`DCCLT_CNV_KEIS83`</u>}

  4:{`DCCLT_CNV_INVSPC` | <u>`DCCLT_CNV_INVERR`</u>}

  5:{`DCCLT_CNV_TAB` | `DCCLT_CNV_NOTAB`}

  6:{`DCCLT_CNV_CNTL` | `DCCLT_CNV_NOCNTL`}

  Description of the specified values

  `DCNOFLAGS`

  Uses the following defaults.

  Uses the EBCDIK code.

  Two-byte spaces remain the same.

  Uses the 1983 version of the KEIS code.

  An error occurs if an invalid code is found.

A tab code is not identified to be single-byte. No shift code is available even for just the preceding or succeeding two-byte code if any.

A control code is not identified to be single-byte. No shift code is available even for just the preceding or succeeding two-byte code if any.

`DCCLT_CNV_EBCDIC`

Uses the EBCDIC code.

`DCCLT_CNV_EBCDIK`

Uses the EBCDIK code.

`DCCLT_CNV_KEIS78`

Uses the 1978 version of the KEIS code .

`DCCLT_CNV_KEIS83`

Uses the 1983 version of the KEIS code.

`DCCLT_CNV_INVSPC`

Converts an invalid code into a space.

`DCCLT_CNV_INVERR`

An error occurs if an invalid code is found.

`DCCLT_CNV_TAB`

Identifies a tab code to be single-byte. A shift code is given to just the preceding or succeeding two-byte code if any.

`DCCLT_CNV_NOTAB`

Does not identify a tab code to be single-byte. No shift code is provided for even just the preceding or succeeding two-byte code if any.

`DCCLT_CNV_CNTL`

Identifies a control code to be single-byte. A shift code is given to just the preceding or succeeding two-byte code if any.

`DCCLT_CNV_NOCNTL`

Does not identify a control code to be single-byte. No shift code is provided for just the preceding or succeeding two-byte code if any.

(4) Arguments for which a value is retuned

- `dest`

  The converted character string is returned.

- `dest_len`

The length of the converted character string is returned.

(5) Return values

Return Value	Value (decimal)	Meaning
`DC_OK`	0	Normal termination
`DCCLTER_INVALID_ARGS`	-2501	A value set for the argument is invalid.
`DCCLTER_NO_BUFS`	-2504	Insufficient memory. The function returns this value also when the control table contains an invalid handle value and when the specified character length covers the first byte of a two-byte code that is contained in the character string to be converted.
`DCCLTER_INVALID_CODE`	-2550	A character string contains an invalid code.
`DCCLTER_OVERFLOW`	-2551	The length of the converted character string exceeds the area prepared by the CUP.

(6) Notes

- The use of this function requires a CommuniNet code mapping table. Before using this function, create a code mapping table using the CommuniNet code mapping utility.
- You cannot use a code mapping table unless you first save the table using the CommuniNet code mapping utility after the installation of CommuniNet. Before using this function, save a code mapping table using the CommuniNet code mapping utility.
- The filename of a CommuniNet code mapping table must be `CMAPEX.TBL`. Store the code mapping table under a Windows directory before using this function.
- The processing of the character code converter does not reflect the changes in the contents of a code mapping table made by the CommuniNet code mapping utility during the use of this function.
- This function does not save error logs and UAP trace information.
- Issue the function for starting character code conversion (`dc_clt_codeconv_open()`) only once for code conversion (`dc_clt_codeconv_exec()`). Do not issue the function for starting character code conversion more than once to prevent memory shortage. If you issue two or more functions, issue one function for terminating character code conversion (`dc_ctl_codeconv_close()`) for each of the issued functions.
- When you specify request to be `DCCLT_EBCKEIS_TO_JISSJIS` and flags to be `DCCLT_CNV_TAB` or `DCCLT_CNV_CNTL`, you need to prepare data that contains

single-byte tab and control codes.

Chapter

5. User Application Program Interface (COBOL Language)

This chapter describes how to create, compile, and link user application programs in COBOL.

In this chapter, COBOL request statements (such as `CBLDCCLS('')`) for calling DLLs are used in explanations. If you use request statements of the normal object library, replace the COBOL request statement names with the corresponding request statement names such as `CBLDCCLT('')`.

This chapter contains the following sections:

5.1 COBOL-UAP creation program features

To use OpenTP1 features, use the COBOL-UAP creation program placed in the TP1/Client library. This section covers features of this program.

When creating a CUP, follow the COBOL/2 or COBOL85 coding specifications. Like an OpenTP1 service user program (SUP), a CUP uses no stubs. Accordingly the user program needs to convert user data codes including code systems and byte orders.

5.1.1 Correspondence between UAPs and facilities

Table 5-1 shows TP1/Client functions and corresponding COBOL-UAP creation programs.

The `CBLDCCLS('')` and other request statements for calling DLLs and those for converting character codes can be used in a multi-thread environment. When you use request statements other than those for converting character codes, we recommend that you use the DLL version of request statements.

Note that all request statements do not always have the DLL versions, depending on the TP1/Client product. For the request statements that have the DLL versions, see the *Release Notes* that comes with the product.

For details on each UAP, see *6 Request Statements Available for TP1/Client (COBOL Language)*.

Table 5-1: TP1/Client functions and corresponding COBOL-UAP creation programs

Function		COBOL-UAP creation program called from the CALL statement
User authentication	Client user authentication request	`CBLDCCLS('CLTIN   ')`
		`CBLDCCLT('CLTIN   ')`
		`CBLDCCLS('EXCLTIN ')`[1]
		`CBLDCCLT('EXCLTIN ')`[1]
	Release of client user authentication	`CBLDCCLS('CLTOUT  ')`
		`CBLDCCLT('CLTOUT  ')`
Remote procedure call	Service response wait time reference	`CBLDCRPS('OPEN    ')`
		`CBLDCRPC('OPEN    ')`
	UAP termination	`CBLDCRPS('CLOSE   ')`

Function		COBOL-UAP creation program called from the CALL statement
		`CBLDCRPC('CLOSE   ')`
	Remote service request	`CBLDCRPS('CALL    ')`
		`CBLDCRPC('CALL    ')`
	Service response wait time update	`CBLDCRPS('SETWATCH')`
		`CBLDCRPC('SETWATCH')`
	UAP startup	`CBLDCRPS('GETWATCH')`
		`CBLDCRPC('GETWATCH')`
Permanent connection	Establishing permanent connection	`CBLDCCLS('CONNECT ')`
		`CBLDCCLT('CONNECT ')`
	Releasing permanent connection	`CBLDCCLS('DISCNCT ')`
		`CBLDCCLT('DISCNCT ')`
	Setting the destination of a request to establish a permanent connection	`CBLDCCLS('STRAPHST')`[2]
		`CBLDCCLT('STRAPHST')`[2]
	Acquiring the destination of a request to establish a permanent connection	`CBLDCCLS('GTRAPHST')`[2]
		`CBLDCCLT('GTRAPHST')`[2]
Transaction control	Transaction startup	`CBLDCTRS('BEGIN   ')`
		`CBLDCTRN('BEGIN   ')`
	Commit in chained mode	`CBLDCTRS('C-COMMIT')`
		`CBLDCTRN('C-COMMIT')`
	Rollback in chained mode	`CBLDCTRS('C-ROLL  ')`
		`CBLDCTRN('C-ROLL  ')`
	Commit in unchained mode	`CBLDCTRS('U-COMMIT')`
		`CBLDCTRN('U-COMMIT')`
	Rollback in unchained mode	`CBLDCTRS('U-ROLL  ')`

Function		COBOL-UAP creation program called from the CALL statement
		`CBLDCTRN('U-ROLL  ')`
	Post information for current transaction	`CBLDCTRS('INFO    ')`
		`CBLDCTRN('INFO    ')`
	Collection of identifiers for current transaction	`CBLDCCLS('GETTRNID')`
		`CBLDCCLT('GETTRNID')`
TCP/IP communication function	Sending messages	`CBLDCCLS('SEND')`
		`CBLDCCLT('SEND')`
		`CBLDCCLS('EXSEND  ')`[1]
		`CBLDCCLT('EXSEND  ')`[1]
	Receiving messages	`CBLDCCLS('RECEIVE')`
		`CBLDCCLT('RECEIVE')`
	Receiving messages (messages receivable even if an error occurs)	`CBLDCCLS('RECEIVE2')`
		`CBLDCCLT('RECEIVE2')`
	Sending assembled messages	`CBLDCCLS('ASMSEND')`
	Receiving assembled messages	`CBLDCCLS('ASMRECV')`
Facility for receiving one-way messages from the server	Receive one-way messages	`CBLDCCLS('NOTIFY  ')`
		`CBLDCCLT('NOTIFY  ')`
		`CBLDCCLS('EXNACPT ')`[1]
		`CBLDCCLT('EXNACPT ')`[1]
	Canceling one-way message wait state	`CBLDCCLS('CANCEL  ')`
		`CBLDCCLT('CANCEL  ')`
		`CBLDCCLS('EXNCANCL')`[1]

Function		COBOL-UAP creation program called from the CALL statement
		`CBLDCCLT('EXNCANCL')`[1]
	Starting reception of one-way messages	`CBLDCCLS('O-NOTIFY')`
		`CBLDCCLT('O-NOTIFY')`
	Terminating reception of one-way messages	`CBLDCCLS('C-NOTIFY')`
		`CBLDCCLT('C-NOTIFY')`
	Receiving a one-way message	`CBLDCCLS('A-NOTIFY')`
		`CBLDCCLT('A-NOTIFY')`
		`CBLDCCLS('EXNCACPT')`[1]
		`CBLDCCLT('EXNCACPT')`[1]
Character code converter (When not using a code mapping table)	Character code converter	`CBLDCUTL('CODECNV ')`
Character code converter (When using a code mapping table)	Starting character code conversion	`CBLDCUTL('CNVOPN  ')`
	Terminating character code conversion	`CBLDCUTL('CNVCLS  ')`
	Executing character code conversion	`CBLDCUTL('CNVEXEC ')`

#1

Use this program when you specify `00000008` for `DCCLTOPTION` in the client environment definition.

#2

Note that you must increase the data area if you specify `00000008` for `DCCLTOPTION` in the client environment definition client.

5.1.2 Format of COBOL-UAP creation program descriptions

When you create a CUP in COBOL, use a CALL statement to call the COBOL-UAP creation program that corresponds to the function in the TP1/Client library.

In *6. Request Statements Available for TP1/Client (COBOL Language)*, each COBOL-UAP creation program is described in a format that consists of the following elements:

Form

Shows the form of calling the COBOL-UAP creation program corresponding to the function in the library using the CALL statement and how to specify areas.

The form is common to COBOL/2 and COBOL85. When you specify a value for the data name, follow the data format of the PICTURE clause indicated here. When necessary values are predefined, the VALUE clause specifies them.

Unless otherwise specified, give specific names to the file name and the data name indicated by the identifier.

Follow COBOL specifications for a character length specified as a data name.

Purpose

Explains the COBOL-UAP creation program function using the following format.

`CBLDC`*XXX*`('`*YYYYYYYY*`')`

YYYYYYYY

Request code

`CBLDC`*XXX*

Name of the COBOL-UAP creation program

Data area where the UAP sets values

Provides data names whose values need to be specified in the data area when the COBOL-UAP creation program is called. The corresponding data in contained in DATA DIVISION. Specify proper values based on respective data descriptions.

Data area for containing the returned values

Provides names of data area in which to contain values returned by OpenTP1, server UAP, and TP1/Client after a `CALL` statement is executed.

Status codes

Describes values returned when the CALL statement is executed. Values are presented in a tabular form. A status code shows whether the COBOL-UAP creation program has executed normally. When an error occurs, the corresponding status code shows its content.

A COBOL status code consists of five digits. It is contained in the first identifier that is specified in the USING clause. The following shows how the CALL statement and the USING clause represent identifiers and a status code.

```
CALL 'program-to-be-called' USING identifier-1 identifier-2 …
```

	Eight characters	Five characters	
identifier-1	Request code	Status code	Area used for the program to be called from CALL statementStatus code

identifier-2	Area used for the program to be called from CALL statement

Notes

Precautions on using respective COBOL-UAP creation programs.

(1) Symbols used for describing the items to be specified

The following table lists the symbols that are used for describing the items to be specified.

Symbol	Description
[ ]	The item enclosed between brackets can be omitted. Example: [:*port-number*] ":*port-number*" can be omitted.
...	This symbol indicates a description is omitted. The item immediately before this symbol can be specified more than once consecutively. Example: *host-name* [:*port-number*][,*host-name* [:*port-number*],...] "*host-name* [:*port-number*]" can be specified more than once consecutively.
~	The item before this symbol conforms to the rule indicated in < > or (( )) after ~.
<character string>	Any character(s)
<unsigned integer>	Numbers 0 to 9
(( ))	The specification range of the specified value is indicated.

5.2 Compiling and linking user application programs

How to compile and link user application programs depends on operating system environments.

5.2.1 Compiling and linking in a UNIX environment

(1) Compile

To create a CUP object file in COBOL, compile the source program using the COBOL compiler. For details of the compilation, see the manual *OpenTP1 Programming Reference COBOL Language*. The following shows how to enter commands for compiling the source program using COBOL85.

Example:

UAP source programs in COBOL

- `cupmain.cbl` (main program)
- `cupfnc1.cbl` (sub-program 1)
- `cupfnc2.cbl` (sub-program 2)

These source programs are compiled as follows.

```
ccbl -C2 -Mw cupmain.cbl
ccbl -C2 cupfncl.cbl
ccbl -C2 cupfnc2.cbl
```

Executing the `ccbl` command generates the following object files.

- `cupmain.o` (object file for the main program)
- `cupfnc1.o` (object file for the sub-program 1)
- `cupfnc2.o` (object file for the sub-program 2)

(2) Link

To create an executable file for CUP, link the following files.

- CUP object file (main program and sub-programs)
- TP1/Client/W library
- COBOL library (or COBOL85 library for the CUP created in COBOL85)

The following shows how to enter commands for linking these files using COBOL85.

Example:

To create a COBOL CUP executable file "example", link these object files as shown below.

- Object file for the main program

  `cupmain.o`

- Object files for the sub-programs

  `cupfnc1.o`, `cupfnc2.o`

```
ccbl -o example cupmain.o cupfnc1.o cupfnc2.o
     -L/usr/lib -lclt
```

Note

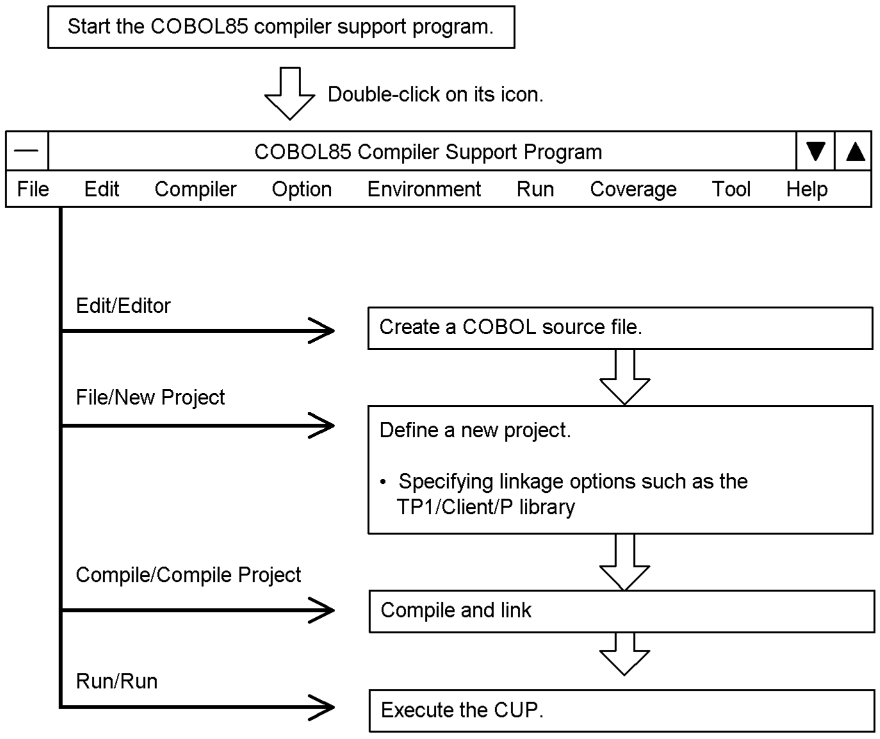
The `-L` option is omissible.

5.2.2 Compiling and linking in a Windows environment

(1) Procedure

The following figure shows how to create a CUP using COBOL85.

Figure 5-1: Creating a new COBOL CUP


Start the COBOL85 compiler support program.
Double-click on its icon.
COBOL85 Compiler Support Program
File
Edit
Compiler
Option
Environment
Run
Coverage
Tool
Help
Edit/Editor
Create a COBOL source file.
File/New Project
Define a new project.
• Specifying linkage options such as the TP1/Client/P library
Compile/Compile Project
Compile and link
Run/Run
Execute the CUP.

(2) Compile, linking, and go

(a) Starting the COBOL85 development

Double-click on the icon for the COBOL85 compiler support program to activate the COBOL85 integrated development environment.

(b) Creating a source program

Select Editor from the Edit menu of the COBOL85 Compiler Support Program window. The screen editor window appears. Enter a source program in this window.

Instead of starting the editor from the COBOL85 Compiler Support Program window, you can use any editor for Windows to create a source program.

(c) Creating a project

Select New Project from the File menu of the COBOL85 Compiler Support Program window.

Based on the instructions in the window, specify the source program's file name, and linkage options.

For the **Import Library/User-Created Library** linkage option, specify `CLTW32.LIB`. If you use the character code converter, also specify `CLTCNV32.LIB`.

(d) Compiling and linking

Choose **Compile Project** from the **Compile** menu of the COBOL85 Compiler Support Program window. The project is automatically compiled and linked.

For linkage, do no specify `/NOI` as a linkage option.

(e) Executing the CUP

Choose **Run** from the **Run** menu of the COBOL85 Compiler Support Program window. After the CUP is created, you can execute it via COBOL85 Compiler Support Program or CBL85R.

5.3 COBOL language template

When creating a UAP in COBOL, you can use COBOL language template to make the Data division coding easy. You can find these templates under the following directories.

- TP1/Client/P

  TP1COBOL under the include directory under the user-specified directory.

  Or, TP1COBOL under the include directory under the directory specified at the most recent installation.

- TP1/Client/W

  `/usr/include/TP1COBOL`

5.3.1 COBOL language template files

The following files are available for COBOL language template.

- `DCCLT.cbl`, `DCCLS.cbl`

  User authentication, TCP/IP communication, and one-way message reception from the server

- `DCRPC.cbl`, `DCRPS.cbl`

  Remote procedure call

- `DCTRN.cbl`, `DCTRS.cbl`

  Transaction control

- `DCUTL.cbl`

  Code conversion (only TP1/Client/P)

5.3.2 Using COBOL language template

When using COBOL language template, you need to modify the following values to match processing of the UAP to be coded.

- Some of data area sizes
- Values placed in data areas

For values placed in data areas, see each function description in Subsection *6. Request Statements Available for TP1/Client (COBOL Language)*. You can use COBOL language template in these two ways.

(1) Using the call function of the text editor

Use a template as follows.

1. Select a proper template from the installation directory.
2. Using the call function of the text editor, cut `DATA DIVISION` from the template, then paste it to the UAP source program.
3. Modify the pasted part so that it can work as a data area for your coding.

(2) Using the COPY statement of COBOL

Use a template as follows.

1. Select a proper template from the installation directory.
2. From the UAP source program, declare COPY using the template file name.
3. Place the template file in the directory the COPY statement can reference. Follow the COBOL implementation when copying files or setting environment variables.
4. Modify the template file so that it can work as a data area for your coding.

5.3.3 Notes on using COBOL language template

- The data area to be modified for UAP processing declares the PICTURE clause length to be (*n*). Compiling a program without changing this value causes an error.
- When you use a template and modify it according to the UAP processing, we recommend you to copy the template from the original directory.

5.4 Example of user application program development

This section shows coding examples to describe how to create CUPs and SPPs when developing UAPs.

5.4.1 Creating CUPs and SPPs

Subsection *3.3.1* provides the configuration for CUP and SPP. The following shows how to create this CUP in COBOL.

```
000010    *
000020    **************************************************
000030    *  CUP sample program                              *
000040    **************************************************
000050    *
000060     IDENTIFICATION DIVISION.
000070     PROGRAM-ID. CUP01.
000080    *
000090    **************************************************
000100    *  Set the data area                               *
000110    **************************************************
000120    *
000130     DATA DIVISION.
000140     WORKING-STORAGE SECTION.
000150     01  DCCLS-CLTIN-ARG.
000160     02  DCCLS-CLTIN-REQUEST      PIC  X(8) VALUE 'CLTIN
'.
000170      02  DCCLS-CLTIN-STATUS-CODE    PIC  X(5).
000180      02  FILLER                     PIC  X(3).
000190      02  DCCLS-CLTIN-FLAGS          PIC S9(9) COMP VALUE
ZERO.
000200      02  DCCLS-CLTIN-T-HOST         PIC  X(64).
000210      02  DCCLS-CLTIN-LOGNAME        PIC  X(16).
000220      02  DCCLS-CLTIN-PASSWD         PIC  X(16).
000230      02  DCCLS-CLTIN-S-HOST         PIC  X(64).
000240      02  DCCLS-CLTIN-HWND           PIC  9(4) COMP.
000250      02  FILLER                     PIC  X(2).
000260      02  DCCLS-CLTIN-CLTID          PIC  9(9) COMP.
000270      02  DCCLS-CLTIN-DEFPATH        PIC  X(256).
000280    *
000290     01  DCCLS-CLTOUT-ARG.
000300     02  DCCLS-CLTOUT-REQUEST     PIC  X(8) VALUE 'CLTOUT
'.
000310      02  DCCLS-CLTOUT-STATUS-CODE   PIC  X(5).
000320      02  FILLER                     PIC  X(3).
000330      02  DCCLS-CLTOUT-FLAGS         PIC S9(9) COMP VALUE
ZERO.
```

```
000340      02  DCCLS-CLTOUT-CLTID          PIC  9(9) COMP.
000350     *
000360     01  DCRPS-OPEN-ARG1.
000370     02  DCRPS-OPEN-REQUEST       PIC  X(8) VALUE 'OPEN
'.
000380      02  DCRPS-OPEN-STATUS-CODE      PIC  X(5).
000390      02  FILLER                      PIC  X(3).
000400      02  DCRPS-OPEN-FLAGS           PIC S9(9) COMP VALUE
ZERO.
000410      02  DCRPS-OPEN-CLTID           PIC  9(9) COMP.
000420     *
000430     01  DCRPS-OPEN-ARG2.
000440      02  FILLER                      PIC  X(1).
000450     *
000460     01  DCRPS-OPEN-ARG3.
000470      02  FILLER                      PIC  X(1).
000480     *
000490     01  DCRPS-CALL-ARG1.
000500     02  DCRPS-CALL-REQUEST       PIC  X(8) VALUE 'CALL
'.
000510      02  DCRPS-CALL-STATUS-CODE      PIC  X(5).
000520      02  FILLER                      PIC  X(3).
000530      02  DCRPS-CALL-FLAGS           PIC S9(9) COMP VALUE
ZERO.
000540      02  DCRPS-CALL-DESCRIPTER       PIC S9(9) COMP.
000550      02  DCRPS-CALL-SVGROUP          PIC  X(32).
000560      02  DCRPS-CALL-SVNAME           PIC  X(32).
000570      02  DCRPS-CALL-CLTID            PIC  9(9) COMP.
000580     *
000590     01  DCRPS-CALL-ARG2.
000600      02  DCRPS-CALL-INDATALEN        PIC S9(9) COMP.
000610      02  DCRPS-CALL-INDATA           PIC  X(512).
000620     *
000630     01  DCRPS-CALL-ARG3.
000640      02  DCRPS-CALL-OUTDATALEN       PIC S9(9) COMP.
000650      02  DCRPS-CALL-OUTDATA         PIC  X(512).
000660     *
000670     01  DCRPS-CLOSE-ARG1.
000680     02  DCRPS-CLOSE-REQUEST      PIC  X(8) VALUE 'CLOSE
'.
000690      02  DCRPS-CLOSE-STATUS-CODE     PIC  X(5).
000700      02  FILLER                      PIC  X(3).
000710      02  DCRPS-CLOSE-FLAGS          PIC S9(9) COMP VALUE
ZERO.
000720      02  DCRPS-CLOSE-CLTID           PIC  9(9) COMP.
000730     *
000740     01  DCRPS-CLOSE-ARG2.
000750      02  FILLER                      PIC  X(1).
```

```
000760     *
000770      01  DCRPS-CLOSE-ARG3.
000780       02  FILLER                      PIC  X(1).
000790     *
000800      77  FOREVER-FLAG     PIC  9      COMP VALUE ZERO.
000810      77  INDATA           PIC  X(512) VALUE SPACE.
000820     *
000830     **************************************************
000840     *  Start CUP                                        *
000850     **************************************************
000860      PROCEDURE DIVISION.
000870      MAIN SECTION.
000880      PROG-START.
000890     *
000900     **************************************************
000910     *  Request client user authentication               *
000920     **************************************************
000930          MOVE 'CLTIN   ' TO DCCLS-CLTIN-REQUEST  IN
DCCLS-CLTIN-ARG.
000940          MOVE ZERO       TO DCCLS-CLTIN-FLAGS    IN
DCCLS-CLTIN-ARG.
000950          MOVE SPACE      TO DCCLS-CLTIN-T-HOST   IN
DCCLS-CLTIN-ARG.
000960          MOVE 'user01'   TO DCCLS-CLTIN-LOGNAME  IN
DCCLS-CLTIN-ARG.
000970          MOVE 'puser01'  TO DCCLS-CLTIN-PASSWD   IN
DCCLS-CLTIN-ARG.
000980          MOVE ZERO       TO DCCLS-CLTIN-HWND     IN
DCCLS-CLTIN-ARG.
000990          MOVE SPACE      TO DCCLS-CLTIN-DEFPATH  IN
DCCLS-CLTIN-ARG.
001000     *
001010     *    ******************************
001020          CALL 'CBLDCCLS' USING DCCLS-CLTIN-ARG.
001030     *    ******************************
001040          IF DCCLS-CLTIN-STATUS-CODE
001050                              IN DCCLS-CLTIN-ARG NOT = '00000'
001060          THEN
001070            DISPLAY 'CUP01: CBLDCCLS(CLTIN) failed. CODE='
001080                 DCCLS-CLTIN-STATUS-CODE IN DCCLS-CLTIN-ARG
001090            GO TO PROG-EXIT
001100          END-IF.
001110     *
001120     **************************************************
001130     *  RPC-OPEN(initialize RPC environment)             *
001140     **************************************************
001150          MOVE 'OPEN    '    TO
001160               DCRPS-OPEN-REQUEST IN DCRPS-OPEN-ARG1.
```

```
001170         MOVE ZERO    TO
001180              DCRPS-OPEN-FLAGS   IN DCRPS-OPEN-ARG1.
001190         MOVE DCCLS-CLTIN-CLTID  IN DCCLS-CLTIN-ARG  TO
001200              DCRPS-OPEN-CLTID   IN DCRPS-OPEN-ARG1.
001210    *
001220    *    *******************************
001230         CALL 'CBLDCRPS' USING DCRPS-OPEN-ARG1
001240                         DCRPS-OPEN-ARG2 DCRPS-OPEN-ARG3.
001250    *    *******************************
001260         IF DCRPS-OPEN-STATUS-CODE IN DCRPS-OPEN-ARG1
001270                                            NOT = '00000'
001280         THEN
001290           DISPLAY 'CUP01: CBLDCRPS(OPEN) failed. CODE='
001300                DCRPS-OPEN-STATUS-CODE IN DCRPS-OPEN-ARG1
001310           GO TO PROG-END
001320         END-IF.
001330    *
001340         PERFORM UNTIL FOREVER-FLAG NOT = ZERO
001350           DISPLAY '****** BBS Menu ******'
001360           DISPLAY 'Read Message .... [1]'
001370                   'Send Message .... [2]'
001380           DISPLAY 'End ............. [9]'
001390           DISPLAY 'Enter a number =>'
001400           ACCEPT INDATA
001410           EVALUATE INDATA
001420           WHEN '1'
001430    *
001440    *    *********************************************
001450    *    * RPC-CALL(execute RPC)                        *
001460    *    *********************************************
001470         MOVE 'CALL    '   TO
001480              DCRPS-CALL-REQUEST  IN DCRPS-CALL-ARG1
001490         MOVE ZERO         TO
001500              DCRPS-CALL-FLAGS    IN DCRPS-CALL-ARG1
001510         MOVE 'spp01'     TO
001520              DCRPS-CALL-SVGROUP  IN DCRPS-CALL-ARG1
001530         MOVE 'get'   TO
001540              DCRPS-CALL-SVNAME   IN DCRPS-CALL-ARG1
001550         MOVE DCCLS-CLTIN-CLTID   IN DCCLS-CLTIN-ARG TO
001560              DCRPS-CALL-CLTID    IN DCRPS-CALL-ARG1
001570         MOVE 'cup01 '     TO
001580              DCRPS-CALL-INDATA   IN DCRPS-CALL-ARG2
001590         MOVE 512      TO
001600              DCRPS-CALL-INDATALEN IN DCRPS-CALL-ARG2
001610         MOVE SPACE    TO
001620              DCRPS-CALL-OUTDATA  IN DCRPS-CALL-ARG3
001630         MOVE 512      TO
001640              DCRPS-CALL-OUTDATALEN IN DCRPS-CALL-ARG3
```

```
001650    *
001660    *
**************************************************
001670         CALL 'CBLDCRPS' USING DCRPS-CALL-ARG1
001680                         DCRPS-CALL-ARG2 DCRPS-CALL-ARG3
001690    *
**************************************************
001700         IF DCRPS-CALL-STATUS-CODE IN DCRPS-CALL-ARG1
001710                                        NOT = '00000'
001720         THEN
001730           DISPLAY 'CUP01: CBLDCRPS(CALL) failed. CODE='
001740                DCRPS-CALL-STATUS-CODE IN DCRPS-CALL-ARG1
001750           GO TO PROG-END
001760         END-IF
001770         DISPLAY 'BBS Contents: ' DCRPS-CALL-OUTDATA
001780                                       IN DCRPS-CALL-ARG3
001790           WHEN '2'
001800         DISPLAY 'Enter your message =>'
001810         ACCEPT INDATA
001820         IF INDATA = SPACE
001830         THEN
001840           MOVE 'No message' TO INDATA
001850         END-IF
001860    *
001870    *    *********************************************
001880    *    * RPC-CALL(execute RPC)                         *
001890    *    *********************************************
001900         MOVE 'CALL    '   TO
001910              DCRPS-CALL-REQUEST  IN DCRPS-CALL-ARG1
001920         MOVE ZERO         TO
001930              DCRPS-CALL-FLAGS    IN DCRPS-CALL-ARG1
001940         MOVE 'spp01'     TO
001950              DCRPS-CALL-SVGROUP  IN DCRPS-CALL-ARG1
001960         MOVE 'put'   TO
001970              DCRPS-CALL-SVNAME   IN DCRPS-CALL-ARG1
001980         MOVE DCCLS-CLTIN-CLTID   IN DCCLS-CLTIN-ARG TO
001990              DCRPS-CALL-CLTID    IN DCRPS-CALL-ARG1
002000         MOVE INDATA     TO
002010              DCRPS-CALL-INDATA   IN DCRPS-CALL-ARG2
002020         MOVE 512      TO
002030              DCRPS-CALL-INDATALEN IN DCRPS-CALL-ARG2
002040         MOVE SPACE    TO
002050              DCRPS-CALL-OUTDATA  IN DCRPS-CALL-ARG3
002060         MOVE 512      TO
002070              DCRPS-CALL-OUTDATALEN IN DCRPS-CALL-ARG3
002080
002090    *
002100    *
```

```
**************************************************
002110         CALL 'CBLDCRPS' USING DCRPS-CALL-ARG1
002120                         DCRPS-CALL-ARG2 DCRPS-CALL-ARG3
002130    *
**************************************************
002140         IF DCRPS-CALL-STATUS-CODE IN DCRPS-CALL-ARG1
002150                                        NOT = '00000'
002160         THEN
002170           DISPLAY 'CUP01: CBLDCRPS(CALL) failed. CODE='
002180                DCRPS-CALL-STATUS-CODE IN DCRPS-CALL-ARG1
002190           GO TO PROG-END
002200         END-IF
002210         DISPLAY DCRPS-CALL-OUTDATA IN DCRPS-CALL-ARG3
002220         WHEN '9'
002230           GO TO PROG-END
002240         WHEN OTHER
002250           CONTINUE
002260         END-EVALUATE
002270         END-PERFORM.
002280     PROG-END.
002290    *
002300    **************************************************
002310    *  RPC-CLOSE(reset RPC environment)                *
002320    **************************************************
002330         MOVE 'CLOSE    '    TO
002340              DCRPS-CLOSE-REQUEST  IN DCRPS-CLOSE-ARG1.
002350         MOVE ZERO       TO
002360              DCRPS-CLOSE-FLAGS    IN DCRPS-CLOSE-ARG1.
002370         MOVE DCCLS-CLTIN-CLTID  IN DCCLS-CLTIN-ARG  TO
002380              DCRPS-CLOSE-CLTID  IN DCRPS-CLOSE-ARG1.
002390    *
002400    *    *******************************
002410         CALL 'CBLDCRPS' USING DCRPS-CLOSE-ARG1.
002420    *    *******************************
002430     PROG-EXIT.
002440         MOVE 'CLTOUT  '  TO
002450              DCCLS-CLTOUT-REQUEST  IN DCCLS-CLTOUT-ARG.
002460         MOVE ZERO     TO
002470              DCCLS-CLTOUT-FLAGS    IN DCCLS-CLTOUT-ARG.
002480         MOVE DCCLS-CLTIN-CLTID   IN DCCLS-CLTIN-ARG   TO
002490              DCCLS-CLTOUT-CLTID  IN DCCLS-CLTOUT-ARG.
002500    *
002510    *    *******************************
002520         CALL 'CBLDCCLS' USING DCCLS-CLTOUT-ARG.
002530    *    *******************************
002540         STOP RUN.
002550    *
002560     MAIN-EXIT SECTION.
```

```
002570         EXIT.
```

5.4.2 Creating a user application program that can run in a multi-thread environment

This subsection describes how to create a COBOL UAP that can run in a multi-thread environment.

(1) Compilation

The following describes how to compile the source program of a UAP written in COBOL. The source program of this UAP requires a thread activation program and the CUP main program.

Write a thread activation program in C, and compile the program by using the `cc` command to create an object file. Write the CUP main program in COBOL, and compile the program by using a COBOL compiler to create the object files.

The following shows the names of example programs used in this subsection, and shows examples of commands for compiling the programs. The examples assume that COBOL85 is used as a COBOL compiler.

- Name of the thread activation program written in C:

  `thdcup_main.c`
- Name of the CUP main program written in COBOL:

  `sample.cbl`

Commands to be entered:

```
xlc_r -c thdcup_main.c
ccbl -c2 -Mt sample.cbl
```

When the above `cc` and `ccbl` commands are executed, the following object files are created:

- `thdcup_main.o` (object file of the thread activation program)
- `sample.o` (object file of the CUP main program)

(2) Linkage

You can use the `ccbl` or `cc` command to create the executable file of a UAP. The following shows how to use the `ccbl` and `cc` commands when creating the executable file of a UAP.

(a) ccbl command

When you use the `ccbl` command to create the executable file of a UAP, you link the following files:

- Object file of the thread activation program
- Object file of the CUP main program
- Library of TP1/Client/W
- Library of COBOL (or libraries of COBOL85 for a CUP created with COBOL85)
- Library of the POSIX thread

The following shows the names of files used in this example, and shows an example of the command for linking the above files by using the `ccbl` command.

- Name of the COBOL CUP executable file to be created:

  `CBL.exe`
- Name of the object file of the thread activation program:

  `thdcup_main.o`
- Name of the object file of the CUP main program:

  `sample.o`

Command to be entered:

```
ccbl -Mt -Mp -o CBL.exe thdcup_main.o sample.o -L/usr/lib -lclt
-lpthread
```

(b) cc command

When you use the `cc` command to create the executable file of a UAP, you link the following files:

- Object file of the thread activation program
- Object file of the CUP main program
- Library of TP1/Client/W
- Library of COBOL (or libraries of COBOL85 for a CUP created with COBOL85)
- Library of the POSIX thread

The following shows the names of files used in this example, and shows an example of the command for linking the above files by using the `cc` command.

- Name of the COBOL CUP executable file to be created:

  `CBL.exe`
- Name of the object file of the thread activation program:

`thdcup_main.o`

- Name of the object file of the CUP main program:

  `sample.o`

Command to be entered:

```
xlc_r -o CBL.exe thdcup_main.o sample.o -L/usr/lib -lclt -L/opt/
HILNGcbl/lib -lcbl85 -lcbl85mp
```

(3) Examples of coding the thread activation program and CUP main program

The following shows examples of the thread activation program (written in C) and the CUP main program (written in COBOL) for a UAP written in COBOL.

(a) Coding example of the thread activation program (written in C)

```
#include <stdio.h>
#include <pthread.h>
#include <sys/errno.h>

#define THDMAX  5

extern void *CUP_THREAD();

main()
{
    int        i;
    int        rc;
    int        exit_value;
    pthread_t  threads[THDMAX];
    struct timeval timeout;

    /* Generates a thread */
    for (i = 1; i < THDMAX; i++) {
        fflush(stdout);
        rc = pthread_create((pthread_t *)&threads[i],
                            NULL,
                            CUP_THREAD,
                            (void *)i);
        if (rc < 0) {
            printf("cup0: pthread_create failed. CODE=%d\n", errno);
        }
    }
}
    /* Waits for the thread to end. */
    for (i = 1; i < THDMAX; i++) {
        rc = pthread_join(threads[i], (void
**)&exit_value);
        if (rc < 0) {
          printf("cup0: pthread_join failed. CODE=%d\n", errno);
        }
    }

}

```

(b) Coding example of the CUP main program (written in COBOL)

```
   *
   ***************************************************
   *  CUP SAMPLE PROGRAM                                   *
   ***************************************************
   *
    IDENTIFICATION DIVISION.
    PROGRAM-ID. CUP_THREAD.
   *
   ***************************************************
   *  DATA AREA SETTINGS                                   *
   ***************************************************
   *
    DATA DIVISION.
    WORKING-STORAGE SECTION.
    01  DCCLS-CLTIN-ARG.
        02  DCCLS-CLTIN-REQUEST        PIC  X(8) VALUE 'CLTIN   '.
        02  DCCLS-CLTIN-STATUS-CODE    PIC  X(5).
        02  FILLER                     PIC  X(3).
        02  DCCLS-CLTIN-FLAGS          PIC S9(9) COMP VALUE ZERO.
        02  DCCLS-CLTIN-T-HOST         PIC  X(64).
        02  DCCLS-CLTIN-LOGNAME        PIC  X(16).
        02  DCCLS-CLTIN-PASSWD         PIC  X(16).
        02  DCCLS-CLTIN-S-HOST         PIC  X(64).
        02  DCCLS-CLTIN-HWND           PIC  9(4) COMP.
        02  FILLER                     PIC  X(2).
        02  DCCLS-CLTIN-CLTID          PIC  9(9) COMP.
        02  DCCLS-CLTIN-DEFPATH        PIC  X(256).
   *
    01  DCCLS-CLTOUT-ARG.
        02  DCCLS-CLTOUT-REQUEST       PIC  X(8) VALUE 'CLTOUT  '.
        02  DCCLS-CLTOUT-STATUS-CODE   PIC  X(5).
        02  FILLER                     PIC  X(3).
        02  DCCLS-CLTOUT-FLAGS         PIC S9(9) COMP VALUE ZERO.
```

```
000340         02  DCCLS-CLTOUT-CLTID          PIC  9(9) COMP.
000350    *
000360     01  DCRPS-OPEN-ARG1.
000370         02  DCRPS-OPEN-REQUEST          PIC  X(8) VALUE
'OPEN    '.
000380         02  DCRPS-OPEN-STATUS-CODE      PIC  X(5).
000390         02  FILLER                      PIC  X(3).
000400         02  DCRPS-OPEN-FLAGS            PIC S9(9) COMP
VALUE ZERO.
000410         02  DCRPS-OPEN-CLTID            PIC  9(9) COMP.
000420    *
000430     01  DCRPS-OPEN-ARG2.
000440         02  FILLER                      PIC  X(1).
000450    *
000460     01  DCRPS-OPEN-ARG3.
000470         02  FILLER                      PIC  X(1).
000480    *
000490     01  DCRPS-CALL-ARG1.
000500         02  DCRPS-CALL-REQUEST          PIC  X(8) VALUE
'CALL    '.
000510         02  DCRPS-CALL-STATUS-CODE      PIC  X(5).
000520         02  FILLER                      PIC  X(3).
000530         02  DCRPS-CALL-FLAGS            PIC S9(9) COMP
VALUE ZERO.
000540         02  DCRPS-CALL-DESCRIPTER       PIC S9(9) COMP.
000550         02  DCRPS-CALL-SVGROUP          PIC  X(32).
000560         02  DCRPS-CALL-SVNAME           PIC  X(32).
000570         02  DCRPS-CALL-CLTID            PIC  9(9) COMP.
000580    *
000590     01  DCRPS-CALL-ARG2.
000600         02  DCRPS-CALL-INDATALEN        PIC S9(9) COMP.
000610         02  DCRPS-CALL-INDATA           PIC  X(512).
000620    *
000630     01  DCRPS-CALL-ARG3.
000640         02  DCRPS-CALL-OUTDATALEN       PIC S9(9) COMP.
000650         02  DCRPS-CALL-OUTDATA          PIC  X(512).
000660    *
000670     01  DCRPS-CLOSE-ARG1.
000680         02  DCRPS-CLOSE-REQUEST         PIC  X(8) VALUE
'CLOSE   '.
000690         02  DCRPS-CLOSE-STATUS-CODE     PIC  X(5).
000700         02  FILLER                      PIC  X(3).
000710         02  DCRPS-CLOSE-FLAGS           PIC S9(9) COMP
VALUE ZERO.
000720         02  DCRPS-CLOSE-CLTID           PIC  9(9) COMP.
000730    *
000740     01  DCRPS-CLOSE-ARG2.
000750         02  FILLER                      PIC  X(1).
```

```
000760    *
000770     01  DCRPS-CLOSE-ARG3.
000780         02  FILLER                     PIC  X(1).
000790    *
000800    **************************************************
000810    *  START OF CUP                                   *
000820    **************************************************
000830     PROCEDURE DIVISION.
000840     MAIN SECTION.
000850     PROG-START.
000860    *
000870    **************************************************
000880    *  CLIENT USER AUTHENTICATION REQUEST             *
000890    **************************************************
000900         MOVE 'CLTIN   '     TO DCCLS-CLTIN-REQUEST IN
DCCLS-CLTIN-ARG.
000910         MOVE ZERO           TO DCCLS-CLTIN-FLAGS   IN
DCCLS-CLTIN-ARG.
000920         MOVE 'host01:10000' TO DCCLS-CLTIN-T-HOST
000930                       IN DCCLS-CLTIN-ARG.
000940         MOVE 'user01'       TO DCCLS-CLTIN-LOGNAME IN
DCCLS-CLTIN-ARG.
000950         MOVE 'puser01'      TO DCCLS-CLTIN-PASSWD  IN
DCCLS-CLTIN-ARG.
000960         MOVE ZERO           TO DCCLS-CLTIN-HWND    IN
DCCLS-CLTIN-ARG.
000970         MOVE SPACE          TO DCCLS-CLTIN-DEFPATH IN
DCCLS-CLTIN-ARG.
000980    *
000990    *    *******************************
001000         CALL 'CBLDCCLS' USING DCCLS-CLTIN-ARG.
001010    *    *******************************
001020       IF DCCLS-CLTIN-STATUS-CODE IN DCCLS-CLTIN-ARG NOT
= '00000'
001030         THEN
001040           DISPLAY 'CUP01: CBLDCCLS(CLTIN) failed. CODE='
001050               DCCLS-CLTIN-STATUS-CODE IN DCCLS-CLTIN-ARG
001060           GO TO PROG-EXIT
001070         END-IF.
001080    *
001090    **************************************************
001100    *  RPC-OPEN(RPC ENVIRONMENT INITIALIZATION)        *
001110    **************************************************
001120         MOVE 'OPEN    '         TO
001130              DCRPS-OPEN-REQUEST IN DCRPS-OPEN-ARG1.
001140         MOVE ZERO               TO DCRPS-OPEN-FLAGS
001150              IN DCRPS-OPEN-ARG1.
001160         MOVE DCCLS-CLTIN-CLTID  IN DCCLS-CLTIN-ARG TO
```

```
001170               DCRPS-OPEN-CLTID   IN DCRPS-OPEN-ARG1.
001180    *
001190    *    ******************************
001200         CALL 'CBLDCRPS' USING DCRPS-OPEN-ARG1
DCRPS-OPEN-ARG2
001210                                          DCRPS-OPEN-ARG3.
001220    *    ******************************
001230        IF DCRPS-OPEN-STATUS-CODE IN DCRPS-OPEN-ARG1 NOT
= '00000'
001240         THEN
001250           DISPLAY 'CUP01: CBLDCRPS(OPEN) failed. CODE='
001260                DCRPS-OPEN-STATUS-CODE IN DCRPS-OPEN-ARG1
001270           GO TO PROG-END
001280         END-IF.
001290    *
001300    *    ********************************************
001310    *    * RPC-CALL(RPC EXECUTION)                  *
001320    *    ********************************************
001330         MOVE 'CALL    '      TO
001340              DCRPS-CALL-REQUEST IN DCRPS-CALL-ARG1.
001350         MOVE ZERO            TO
001360              DCRPS-CALL-FLAGS IN DCRPS-CALL-ARG1.
001370         MOVE 'spp01'       TO
001380              DCRPS-CALL-SVGROUP IN DCRPS-CALL-ARG1.
001390         MOVE 'svr01'     TO
001400              DCRPS-CALL-SVNAME     IN DCRPS-CALL-ARG1.
001410         MOVE DCCLS-CLTIN-CLTID  IN DCCLS-CLTIN-ARG TO
001420              DCRPS-CALL-CLTID   IN DCRPS-CALL-ARG1.
001430         MOVE 'HELLO SPP !! ' TO
001440              DCRPS-CALL-INDATA     IN DCRPS-CALL-ARG2.
001450         MOVE 512             TO
001460              DCRPS-CALL-INDATALEN  IN DCRPS-CALL-ARG2.
001470         MOVE SPACE           TO
001480              DCRPS-CALL-OUTDATA    IN DCRPS-CALL-ARG3.
001490         MOVE 512             TO
001500              DCRPS-CALL-OUTDATALEN IN DCRPS-CALL-ARG3.
001510    *
001520    *
**************************************************
001530         CALL 'CBLDCRPS' USING DCRPS-CALL-ARG1
DCRPS-CALL-ARG2
001540                               DCRPS-CALL-ARG3
001550    *
**************************************************
001560        IF DCRPS-CALL-STATUS-CODE IN DCRPS-CALL-ARG1 NOT
= '00000'
001570         THEN
001580           DISPLAY 'CUP01: CBLDCRPS(CALL) failed.'
```

```
                   'CODE=' DCRPS-CALL-STATUS-CODE IN DCRPS-CALL-ARG1
           GO TO PROG-END
         END-IF.
     PROG-END.
    *
    **************************************************
   *  RPC-CLOSE(RPC ENVIRONMENT RELEASE)                 *
    **************************************************
         MOVE 'CLOSE   ' TO DCRPS-CLOSE-REQUEST IN DCRPS-CLOSE-ARG1.
         MOVE ZERO       TO DCRPS-CLOSE-FLAGS   IN DCRPS-CLOSE-ARG1.
         MOVE DCCLS-CLTIN-CLTID  IN DCCLS-CLTIN-ARG TO
              DCRPS-CLOSE-CLTID  IN DCRPS-CLOSE-ARG1.
    *
    *    *******************************
         CALL 'CBLDCRPS' USING DCRPS-CLOSE-ARG1 DCRPS-CLOSE-ARG2
                                        DCRPS-CLOSE-ARG3.
    *    *******************************
     PROG-EXIT.
         MOVE 'CLTOUT  '  TO DCCLS-CLTOUT-REQUEST IN DCCLS-CLTOUT-ARG.
         MOVE ZERO        TO DCCLS-CLTOUT-FLAGS   IN DCCLS-CLTOUT-ARG.
         MOVE DCCLS-CLTIN-CLTID  IN DCCLS-CLTIN-ARG TO
              DCCLS-CLTOUT-CLTID IN DCCLS-CLTOUT-ARG.
    *
    *    *******************************
         CALL 'CBLDCCLS' USING DCCLS-CLTOUT-ARG.
    *    *******************************
         STOP RUN.
    *
```

Chapter

6. Request Statements Available for TP1/Client (COBOL Language)

6.1 Notes on using request statements

When you use request statements, we recommend that you use request statements that are suitable for a multi-thread environment even if you are using a single-thread environment.

In a multi-thread environment, do not use request statements that are suitable for a single-thread environment.

6.2 User authentication

6.2.1 CBLDCCLS('CLTIN   ') - Client user authentication request

(1) Form

(a) In a multi-thread environment

■ **PROCEDURE DIVISION**

```
CALL 'CBLDCCLS' USING identifier-1
```

■ **DATA DIVISION**

```
01 identifier-1.
 02 data-name-A PIC X(8)  VALUE 'CLTIN   '.
 02 data-name-B PIC X(5).
 02 FILLER     PIC X(3).
 02 data-name-C PIC S9(9) COMP VALUE ZERO.
 02 data-name-D PIC X(64).
 02 data-name-E PIC X(16).
 02 data-name-F PIC X(16).
 02 data-name-G PIC X(64).
 02 FILLER     PIC 9(4) COMP.
 02 FILLER     PIC X(2).
 02 data-name-I PIC 9(9) COMP.
 02 data-name-J PIC X(256).
```

(b) In a single-thread environment

■ **PROCEDURE DIVISION**

```
CALL 'CBLDCCLT' USING identifier-1
```

■ **DATA DIVISION**

```
01 identifier-1.
   02 data-name-A PIC X(8)  VALUE 'CLTIN   '.
   02 data-name-B PIC X(5).
   02 FILLER     PIC X(3).
   02 data-name-C PIC S9(9) COMP VALUE ZERO.
   02 data-name-D PIC X(64).
   02 data-name-E PIC X(16).
```

```
02 data-name-F PIC X(16).
02 data-name-G PIC X(64).
```

(2) Purpose

Requests authentication of the client user specified with the login name corresponding to the specified TP1/Server to be used as a gateway.

Always execute `CBLDCCLS('CLTIN    ')` even when you suppress user authentication.

(3) Data area where the UAP sets values

- *data-name-A*

  Set `VALUE 'CLTIN△△△'` as a request code for the client user authentication request.

- *data-name-C*

  Set `-2147483648` to suppress user authentication for using the remote API facility. Set `0` not to suppress user authentication.

- *data-name-D*

  Set the host name and port number of TP1/Server you want to use as a gateway when issuing an authentication request.

  You can specify more than one TP1/Server used as a gateway separated by a comma (`,`). You can also specify an IP address in decimal dot notation for the host name.

  Format:

  host-computer-name[:*port-number*][,*host-computer-name*[:*port-number*],...]

  host-computer-name~<character string>

  port-number~<unsigned integer>((5001-65535))

  Do not place a null character (space or tab) except after the separator (,).

  When the port number is omitted, the system assumes the value for `DCNAMPORT` in the client environment definition.

  When you have specified more than one TP1/Server in *data-name-D* and an error is detected in the TP1/Server being used as a gateway, system operation depends on the specification of `DCHOSTSELECT` in the client environment definition. If `N` is specified, the system attempts to replace the failed node by referencing the next TP1/Server of the currently used TP1/Server. If `Y` is specified, the system selects a TP1/Server at random (except for the TP1/Server in which the error was detected) and attempts to replace the failed node.

When *data-name-D* starts with a blank, the program references `DCHOST` in the client environment definition. If *data-name-D* starts with a blank and `DCHOST` is not set, a broadcast is performed to determine the target host computer.

To perform a broadcast in TP1/Client/P, you must specify the broadcast address in the `hosts` file (the host name must be `broadcast`). If the host name is not specified, `CBLDCCLS('EXCLTIN ')`returns an error with status code 02518.

Terminate the character string with a blank.

- *data-name-E*

  Store the client user's login name.

  Terminate the character string with a blank.

- *data-name-F*

  Store a password for the login name specified with *data-name-E*. If no password is available, place a blank at the beginning of *data-name-F*.

  Terminate the character string with a blank.

- *data-name-J*

  Specify the path name of the client environment definition file. The path name must be specified with the full path or with a relative path from the current drive and the current directory. The following shows the order in which files are loaded when the path name is specified.

  - In TP1/Client/P

    Client environment definition files are loaded in the following order:

    1. The `BETRAN.INI` file in the Windows directory

    2. The client environment definition file specified in *data-name-J* argument

    The definitions in both the client environment definition file and the `BETRAN.INI` file take effect.

    If the same definition is specified in each file with a different value, the value specified in the client environment definition file takes effect.

    If neither the client environment definition file nor the `BETRAN.INI` file contains the necessary specification, TP1/Client/P uses the defaults.

  - In TP1/Client/W

    All definitions specified in the environment variables will be invalid. TP1/Client/W uses the defaults for definitions that are not specified in the client environment definition file specified in *data-name-J*.

You can omit the path name by specifying a blank at the beginning of *data-name-J*. The following describes the operation when the path name is omitted.

- In TP1/Client/P

  TP1/Client/P uses the `BETRAN.INI` file in the Windows directory as the client environment definition file. If the `BETRAN.INI` file does not exist or if the contents of the definition file are invalid, TP1/Client/P uses the defaults.
- In TP1/Client/W

  TP1/Client/W uses the values specified in the environment variables. If an environment variable is not specified, TP1/Client/W uses the default.

The following describes operation when the client environment definition file specified in *data-name-J* does not exist or when the contents of the definition file are invalid.

- In TP1/Client/P

  TP1/Client/P uses the `BETRAN.INI` file in the Windows directory as the client environment definition file. If the `BETRAN.INI` file does not exist or if the contents of the definition file are invalid, TP1/Client/P uses the defaults.
- In TP1/Client/W

  TP1/Client/W uses the defaults. The values specified in the environment variables will be invalid.

(4) Data area for which a value is returned

- *data-name-B*

  5-digit status code.
- *data-name-G*

  The host name (or IP address in decimal-dot notation) of the server that actually performed user authentication. Nothing is returned if you suppress user authentication.

  The stored host computer name ends with a blank.
- *data-name-I*

  A client ID is set when client user authentication is completed successfully. Do not destroy the client ID before `CBLDCCLS('CLTOUT  ')` is executed.

(5) Status codes

Status code	Meaning
00000	Normal termination
02501	Invalid value for the data name. The request code (*data-name-A*) may be invalid.
02502	`CBLDCCLT('CLTIN   ')` has already been executed. This status code is not returned if `CBLDCCLS('CLTIN   ')` is executed.
02503	The communication path initialization failed. Alternatively the client environment definition is specified incorrectly.
02504	A necessary amount of buffer could not be allocated.
02506	Network error
02515	OpenTP1 is inactive for the node that has the specified service.
02518	System error
02527	This status code is returned because of one of the following reasons: The specified *data-name-E* is not registered in the target host. The password does not match. The OpenTP1 server may not support user authentication. `client_uid_check` is specified incorrectly in the system common definition.
02547	The specified port number is in use. Alternatively, port numbers that can be assigned automatically by the operating system are insufficient.

6.2.2 CBLDCCLS('EXCLTIN ') - Client user authentication request (for an extended host name)

(1) Form

(a) In a multi-thread environment

■ **PROCEDURE DIVISION**

```
CALL 'CBLDCCLS' USING identifier-1 identifier-2 identifier-3
```

■ **DATA DIVISION**

```
01 identifier-1.
   02 data-name-A PIC X(8)  VALUE 'EXCLTIN '.
   02 data-name-B PIC X(5).
   02 FILLER     PIC X(3).
   02 data-name-C PIC S9(9) COMP VALUE ZERO.
   02 data-name-D PIC X(16).
```

```
      02 data-name-E PIC X(16).
      02 data-name-F PIC X(n).
   01 identifier-2.
      02 FILLER     PIC X(9) COMP.
      02 data-name-G PIC X(n).
   01 identifier-3.
      02 data-name-H PIC 9(9) COMP.
      02 FILLER     PIC 9(4) COMP.
      02 FILLER     PIC X(2).
      02 data-name-I  PIC X(n).
```

(b) In a single-thread environment

■ **PROCEDURE DIVISION**

```
CALL 'CBLDCCLT' USING identifier-1 identifier-2
```

■ **DATA DIVISION**

```
01 identifier-1.
   02 data-name-A PIC X(8)  VALUE 'EXCLTIN '.
   02 data-name-B PIC X(5).
   02 FILLER     PIC X(3).
   02 data-name-C PIC S9(9) COMP VALUE ZERO.
   02 data-name-D PIC X(16).
   02 data-name-E PIC X(16).
   02 data-name-F PIC X(n).
01 identifier-2.
   02 FILLER     PIC X(9) COMP.
   02 data-name-G  PIC X(n).
```

(2) Purpose

Requests authentication of the client user specified with the login name corresponding to the specified TP1/Server to be used as a gateway.

Always execute `CBLDCCLS('EXCLTIN ')`, even if you have suppressed user authentication.

Use this function when using the host name extension function.

(3) Data area where the UAP sets values

- *data-name-A*

  Set `VALUE 'EXCLTIN∆'` as a request code for the client user authentication request.

- *data-name-C*

  Set `-2147483648` to suppress user authentication for using the remote API facility. Set `0` not to suppress user authentication.

- *data-name-D*

  Set the client user's login name.

  Terminate the character string with a blank.

- *data-name-E*

  Set a password for the login name specified with *data-name-D*. If no password is available, place a blank at the beginning of *data-name-E*.

  Terminate the character string with a blank.

- *data-name-F*

  Set the host name and port number of the TP1/Server you want to use as a gateway when issuing an authentication request. You can specify more than one TP1/ Server for use as a gateway by separating them with a comma (`,`).

  You can also specify an IP address in decimal dot notation for the host name.

  Format

  > *host-name*[:*port-number*][,*host-name*[:*port-number*],...]
  >
  > *host-name*~<character string>
  >
  > *port-number*~<unsigned integer>((5001-65535))

  You can specify a maximum of 63# characters for the host name. When specifying multiple host names, you can specify a maximum of 255# characters, including port numbers, in *data-name-F*. Terminate the character string with a blank.

  Do not place a blank character (space or tab) except after the separator (,). When the port number is omitted, the system assumes the value for `DCNAMPORT` in the client environment definition.

  When you have specified more than one TP1/Server in *data-name-F* and an error is detected in the TP1/Server being used as a gateway, system operation depends on the specification of `DCHOSTSELECT` in the client environment definition. If `N` is specified, the system attempts to replace the failed node by referencing the next TP1/Server of the currently used TP1/Server. If `Y` is specified, the system selects a TP1/Server at random (except for the TP1/Server in which the error was detected) and attempts to replace the failed node.

When *data-name-F* starts with a blank, the program references `DCHOST` in the client environment definition.

If *data-name-F* starts with a blank and `DCHOST` is not set, a broadcast is performed to determine the TP1/Server to be used as a gateway.

To perform a broadcast in TP1/Client/P, you must specify the broadcast address in the `hosts` file (the host name must be `broadcast`). If the host name is not specified, `CBLDCCLS('EXCLTIN ')` returns an error with status code 02518.

Terminate the character string with a blank.

# If you specify `00000008` for `DCCLTOPTION` in the client environment definition, you can specify a maximum of 255 characters for the host name. When specifying multiple host names, you can specify a maximum of 1023 characters, including port numbers, in *data-name-F*.

- *data-name-G*

  Specify an area of 64 bytes# or more for storing the host name of the server that actually performed user authentication.

  # This area must be larger than 255 bytes if you specify `00000008` for `DCCLTOPTION` in the client environment definition.

- *data-name-I*

  Specify the path name of the client environment definition file. The path name must be specified with the full path or with a relative path from the current drive and the current directory. The following shows the order in which files are loaded when the path name is specified.

  - In TP1/Client/P

    Client environment definition files are loaded in the following order:

    1. The `BETRAN.INI` file in the Windows directory

    2. The client environment definition file specified in *data-name-I* argument

    The definitions in both the client environment definition file and the `BETRAN.INI` file take effect.

    If the same definition is specified in each file with a different value, the value specified in the client environment definition file takes effect.

    If neither the client environment definition file nor the `BETRAN.INI` file contains the necessary specification, TP1/Client/P uses the defaults.

  - In TP1/Client/W

    All definitions specified in the environment variables will be invalid. TP1/Client/W uses the defaults for definitions that are not specified in the client environment definition file specified in *data-name-I*.

You can omit the path name by specifying a blank at the beginning of *data-name-I*. The following describes the operation when the path name is omitted.

- In TP1/Client/P

  TP1/Client/P uses the `BETRAN.INI` file in the Windows directory as the client environment definition file. If the `BETRAN.INI` file does not exist or if the contents of the definition file are invalid, TP1/Client/P uses the defaults.

- In TP1/Client/W

  TP1/Client/W uses the values specified in the environment variables. If an environment variable is not specified, TP1/Client/W uses the default.

The following describes operation when the client environment definition file specified in *data-name-I* does not exist or when the contents of the definition file are invalid.

- In TP1/Client/P

  TP1/Client/P uses the `BETRAN.INI` file in the Windows directory as the client environment definition file. If the `BETRAN.INI` file does not exist or if the contents of the definition file are invalid, TP1/Client/P uses the defaults.

- In TP1/Client/W

  TP1/Client/W uses the defaults. The values specified in the environment variables will be invalid.

(4) Data area for which a value is returned

- *data-name-B*

  5-digit status code.

- *data-name-G*

  The host name (or IP address in decimal-dot notation) of the server that actually performed user authentication. Nothing is returned if you suppress user authentication.

  The stored host name ends with a blank.

- *data-name-H*

  A client ID is set when client user authentication is completed successfully. Do not destroy the client ID before `CBLDCCLS('CLTOUT  ')` is executed.

(5) Status codes

Status code	Meaning
00000	Normal termination
02501	Invalid value for the data name. The request code (*data-name-A*) may be invalid.
02502	`CBLDCCLT('EXCLTIN ')` has already been executed. This status code is not returned if `CBLDCCLS('EXCLTIN ')` is executed.
02503	An attempt to initialize the communication path failed. Alternatively, the client environment definition is specified incorrectly.
02504	A necessary amount of buffer could not be allocated.
02506	Network error
02515	OpenTP1 is not running on the node that has the specified service.
02518	System error
02527	This status code is returned because of one of the following reasons: The specified login name (*data-name-D*) is not registered in the target host. The password (*data-name-E*) does not match. The OpenTP1 server may not support user authentication. Check whether `client_uid_check` is specified correctly in the system common definition.
02547	The specified port number is in use. Alternatively, port numbers that can be assigned automatically by the operating system are insufficient.

6.2.3 CBLDCCLS('CLTOUT ') - Release of client user authentication

(1) Form

(a) In a multi-thread environment

■ **PROCEDURE DIVISION**

```
CALL 'CBLDCCLS' USING identifier-1
```

■ **DATA DIVISION**

```
01 identifier-1.
   02 data-name-A PIC X(8)  VALUE 'CLTOUT '.
   02 data-name-B PIC X(5).
   02 FILLER     PIC X(3).
   02 data-name-C PIC S9(9) COMP VALUE ZERO.
   02 data-name-D PIC 9(9) COMP.
```

(b) In a single-thread environment

■ **PROCEDURE DIVISION**

```
CALL 'CBLDCCLT' USING identifier-1
```

■ **DATA DIVISION**

```
01 identifier-1.
   02 data-name-A PIC X(8)  VALUE 'CLTOUT '.
   02 data-name-B PIC X(5).
   02 FILLER     PIC X(3).
   02 data-name-C PIC S9(9) COMP VALUE ZERO.
```

(2) Purpose

Releases the client user authentication and rejects OpenTP1 services afterwards.

Be sure to execute `CBLDCCLS('CLTOUT  ')` before termination of a CUP. When executed, `CBLDCCLS('CLTOUT  ')` must be paired with `CBLDCCLS('CLTIN   ')`.

(3) Data area where the UAP sets values

■ *data-name-A*

Set `VALUE 'CLTOUT△△'` as a request code for releasing the client user authentication.

■ *data-name-C*

Set `0`.

■ *data-name-D*

Specify the client ID received with `CBLDCCLS('CLTIN   ')` or `CBLDCCLS('EXCLTIN ')`.

(4) Data area where OpenTP1 returns values

■ *data-name-B*

5-digit status code.

(5) Status codes

Status code	Meaning
00000	Normal termination

Status code	Meaning
02501	The request code (*data-name-A*) may be invalid.

6.3 Remote procedure calls

6.3.1 CBLDCRPS('OPEN   ') - UAP startup

(1) Form

(a) In a multi-thread environment

■ **PROCEDURE DIVISION**

```
CALL 'CBLDCRPC' USING identifier-1
```

■ **DATA DIVISION**

```
01 identifier-1.
   02 data-name-A PIC X(8)  VALUE 'OPEN    '.
   02 data-name-B PIC X(5).
   02 FILLER      PIC X(3).
   02 data-name-C PIC S9(9) COMP VALUE ZERO.
```

(b) In a single-thread environment

■ **PROCEDURE DIVISION**

```
CALL 'CBLDCRPS' USING identifier-1  identifier-2  identifier-3
```

■ **DATA DIVISION**

```
01 identifier-1.
   02 data-name-A PIC X(8)  VALUE 'OPEN    '.
   02 data-name-B PIC X(5).
   02 FILLER      PIC X(3).
   02 data-name-C PIC S9(9) COMP VALUE ZERO.
   02 data-name-D PIC 9(9)  COMP.
01 identifier-2.
   02 FILLER      PIC X(1).
01 identifier-3.
   02 FILLER      PIC X(1).
```

(2) Purpose

Initializes the environment for calling the OpenTP1 SPP or using the TCP/IP communication facility.

Always execute `CBLDCRPS('OPEN      ')` before executing any RPC program or transaction control program.

(3) Data area where the UAP sets values

- *data-name-A*

  Set `VALUE 'OPEN△△△△'` as a request code for starting the UAP.

- *data-name-C*

  Specify the environment to be initialized. You can specify one of the following environments:

  `0`: Environment for calling the SPP

  `4`: Environment for sending one-way messages

  `8`: Environment for receiving one-way messages

  `16`: Environment for sending and receiving messages

  Specify `4`, `8`, or `16` to use the TCP/IP communication facility. The RPC facility is also available, when you specify `4`, `8`, or `16`.

- *data-name-D*

  Specify the client ID received with `CBLDCCLS('CLTIN    ')` or `CBLDCCLS('EXCLTIN ')`.

(4) Data area for which a value is returned

- *data-name-B*

  5-digit status code.

(5) Status codes

Status code	Meaning
00000	Normal termination
02401	Invalid value for the data name. The request code (*data-name-A*) may be invalid.
02402	`CBLDCRPS('OPEN      ')` has already been executed.
02403	One of the following errors occurs. • Initialization failed. • No user authentication is performed. • The client environment definition is specified invalidly.
02415	OpenTP1 is inactive for the node corresponding to the specified service.
02447	The specified port number is in use.

Status code	Meaning
02544	The client ID specified for *data-name-D* differs from the one received with `CBLDCCLS('CLTIN')` or `CBLDCCLS('EXCLTIN ')`.

(6) Notes

- Just after you execute `CBLDCRPS('CLOSE   ')`, you cannot execute `CBLDCRPS('OPEN    ')` whose *data-name-C* is 8 under the following condition.

  After execution of `CBLDCRPS('OPEN    ')` whose *data-name-C* is 8, `CBLDCCLS('RECEIVE ')` is executed to receive messages. Before the remote system releases the connection, the CUP releases it by executing `CBLDCRPS('CLOSE   ')`.

In this case, wait 15 to 20 seconds, then execute `CBLDCRPS('OPEN    ')`.

6.3.2 CBLDCRPS('CLOSE   ') - UAP termination

(1) Form

(a) In a multi-thread environment

■ **PROCEDURE DIVISION**

```
CALL 'CBLDCRPS' USING identifier-1  identifier-2  identifier-3
```

■ **DATA DIVISION**

```
01 identifier-1.
   02 data-name-A PIC X(8)  VALUE 'CLOSE   '.
   02 data-name-B PIC X(5).
   02 FILLER     PIC X(3).
   02 data-name-C PIC S9(9) COMP VALUE ZERO.
   02 data-name-D PIC 9(9)  COMP.
01 identifier-2.
   02 FILLER     PIC X(1).
01 identifier-3.
   02 FILLER     PIC X(1).
```

(b) In a single-thread environment

■ **PROCEDURE DIVISION**

```
CALL 'CBLDCRPC' USING identifier-1
```

■ **DATA DIVISION**

```
01 identifier-1.
   02 data-name-A PIC X(8)  VALUE 'CLOSE   '.
   02 data-name-B PIC X(5).
   02 FILLER     PIC X(3).
   02 data-name-C PIC S9(9) COMP VALUE ZERO.
```

(2) Purpose

Releases the environment for calling the OpenTP1 SPP or using the TCP/IP communication facility.

When executed, `CBLDCRPS('CLOSE   ')` must be paired with `CBLDCRPS('OPEN    ')`. The following programs are available after execution of `CBLDCRPS('CLOSE   ')`.

- `CBLDCRPS('OPEN    ')`
- `CBLDCCLS('CLTOUT  ')`

(3) Data area where the UAP sets values

■ *data-name-A*

Set `VALUE 'CLOSE△△△'` as the request code for indicating termination of the UAP.

■ *data-name-C*

Set 0.

■ *data-name-D*

Specify the client ID received with `CBLDCCLS('CLTIN   ')` or `CBLDCCLS('EXCLTIN ')`.

(4) Data area for which a value is returned

■ *data-name-B*

5-digit status code.

(5) Status codes

Status code	Meaning
00000	Normal termination
02401	The request code (*data-name-A*) may be invalid.

6.3.3 CBLDCRPS('CALL   ') - Remote service request

(1) Form

(a) In a multi-thread environment

■ **PROCEDURE DIVISION**

```
CALL 'CBLDCRPS' USING identifier-1 identifier-2 identifier-3
```

■ **DATA DIVISION**

```
01 identifier-1.
   02 data-name-A PIC X(8)  VALUE 'CALL    '.
   02 data-name-B PIC X(5).
   02 FILLER     PIC X(3).
   02 data-name-C PIC S9(9) COMP VALUE ZERO.
   02 data-name-D PIC S9(9) COMP.
   02 data-name-E PIC X(32).
   02 data-name-F PIC X(32).
   02 data-name-G PIC 9(9)  COMP.
01 identifier-2.
   02 data-name-H PIC S9(9) COMP.
   02 data-name-I PIC X(n).
01 identifier-3.
   02 data-name-J PIC S9(9) COMP.
   02 data-name-K PIC X(n).
```

(b) In a single-thread environment

■ **PROCEDURE DIVISION**

```
CALL 'CBLDCRPC' USING identifier-1 identifier-2 identifier-3
```

■ **DATA DIVISION**

```
01 identifier-1.
   02 data-name-A PIC X(8)  VALUE 'CALL    '.
   02 data-name-B PIC X(5).
   02 FILLER     PIC X(3).
   02 data-name-C PIC S9(9) COMP VALUE ZERO.
   02 data-name-D PIC S9(9) COMP.
   02 data-name-E PIC X(32).
   02 data-name-F PIC X(32).
```

```
01 identifier-2.
   02 data-name-H PIC S9(9) COMP.
   02 data-name-I PIC X(n).
01 identifier-3.
   02 data-name-J PIC S9(9) COMP.
   02 data-name-K PIC X(n).
```

(2) Purpose

Requests an SPP service by calling the service program that matches the service group name and the service name and receiving its response.

OpenTP1 must be active for the node corresponding to the server UAP for which the service is requested. If OpenTP1 is inactive due to a startup procedure, for example, the `CBLDCRPS('CALL    ')` program returns an error with status code 02406, 02415, or 02420. The program returns an error with status code 02412 when it is executed but the target service group is shut down. The program returns an error with status code 02413, 02412, or 02410 when it is executed but the target service group is terminating or has been terminated due to a `dcsvstop` command, for example. Which status code returns depends on the timing when the `CBLDCRPS('CALL    ')` program was executed.

A socket-receiving type server concurrently controls messages by specifying `max_socket_msg` and `max_socket_msglen` in the user service definition. This may prevent receiving service requests. In this case, `CBLDCRPS('CALL    ')` returns an error with status code 02456. When this value returns, wait a while, then reexecute the CUP. You may succeed in the service request.

For the normal communication mode, specify the host name and port number of the XDM/DCCM3 logical terminal in `DCCLTSERVICEGROUPLIST` in the client environment definition, and then execute `CBLDCRPS('CALL    ')`.

(a) Values passed to server UAP

The CUP allocates an area (*data-name-K*) for responding to the service program. It also specifies the following values for `CBLDCRPS('CALL    ')`.

- Input parameter (*data-name-I*)
- Input parameter length (*data-name-H*)
- Response length (*data-name-J*)

These values are same as those specified for `CBLDCRPS('CALL    ')` in the CUP and are passed (and unchanged) to the service program. When you call services of a service program that returns no response, the response length, if specified, is ignored. The maximum values for the input parameter length and response length are defined with `DCRPC_MAX_MESSAGE_SIZE`# in the `dcvrpc.h` header file.

If you specify 2 or a larger value for `DCCLTRPCMAXMSGSIZE` in the client environment definition, the value you specify is used rather than the value of `DCRPC_MAX_MESSAGE_SIZE` (1 megabyte).

(b) Values returned from server UAP

You can reference the following values after termination of the service program.

- Service program response (*data-name-K*)
- Service program response length (*data-name-J*)

Data-name-J shows the length of a response actually returned from the service program.

After `CBLDCRPS('CALL    ')` returns, a synchronous-response type RPC (*data-name-C* set to 0) can reference *data-name-K* and *data-name-J*. A no-response type RPC (*data-name-C* set to 1) cannot reference these values. When `CBLDCRPS('CALL    ')` returns an error, *data-name-K* and *data-name-J* cannot be referenced.

When the returned response exceeds the response area (*data-name-K*) allocated by the CUP, the program returns an error with status code 02409.

(3) Data area where the UAP sets values

■ *data-name-A*

Set `VALUE'CALL△△△△'` as a request code for the remote service request.

■ *data-name-C*

Specify the RPC type as follows.

`0`: Synchronous-response type RPC

`1`: No-response type RPC

`4`: Chained RPC

When you specify `0` or `4` for *data-name-C*, `CBLDCRPS('CALL    ')` does not return until a response returns or a response wait timeout error occurs based on the `DCWATCHTIM` value in the client environment definition. When the service-requested SPP aborts, the program immediately returns an error. The returned status code depends on the response wait time specified with `DCWATCHTIM` as follows.

`DCWATCHTIM = 1-65535:02407`

`DCWATCHTIM = 0` (infinite wait)`: 02414`

You cannot specify a response wait time for each service-requested service program or for each service request.

When you specify `1` for *data-name-C*, the system assumes that the requested service does not return a response. Therefore, `CBLDCRPS('CALL    ')` immediately returns without waiting for the service to terminate. When *data-name-C* = 1, you cannot reference the response (*data-name-K*) and the response length (*data-name-J*). The CUP cannot determine if the service program was executed.

You can change an RPC issued from the transaction to a service request that is not a transaction. To do this, specify `32` for the parameter that indicates the RPC type. The service request for the corresponding `CBLDCRPS('CALL    ')` program will be a non-transaction service request.

`32`: Synchronous-response type RPC

`33`: No-response type RPC

`36`: Chained RPC

If you specify `4` either not in a transaction or when the permanent connection is not being established, the program returns an error with status code 02401.

- *data-name-E*

  Set a service group name using up to 31 ASCII characters ending with a blank.

- *data-name-F*

  Set a service name using up to 31 ASCII characters ending with a blank.

- *data-name-G*

  Specify the client ID received with `CBLDCCLS('CLTIN   ')` or `CBLDCCLS('EXCLTIN ')`.

- *data-name-H*

  Set the input parameter length (*data-name-I* length) except the length of *data-name-H* itself. Available values range from 1 to `DCRPC_MAX_MESSAGE_SIZE`#.

  # If you specify 2 or a larger value for `DCCLTRPCMAXMSGSIZE` in the client environment definition, the value you specify is used rather than the value of `DCRPC_MAX_MESSAGE_SIZE` (1 megabyte).

- *data-name-I*

  Set an input parameter.

- *data-name-J*

  Set the response storage area length (*data-name-K* length) except the length of *data-name-J* itself. Available values range from 1 to `DCRPC_MAX_MESSAGE_SIZE`#.

If you specify 2 or a larger value for `DCCLTRPCMAXMSGSIZE` in the client environment definition, the value you specify is used rather than the value of `DCRPC_MAX_MESSAGE_SIZE` (1 megabyte).

- *data-name-K*

  Specify the area for storing the response. This area must be larger than the length specified for *data-name-J*.

(4) Data area for which a value is returned

- *data-name-B*

  5-digit status code.

- *data-name-D*

  Area used for OpenTP1.

- *data-name-J*

  Response length (*data-name-K* length). This value is not returned when `DCRPC_NOREPLY` is specified in *data-name-C*.

- *data-name-K*

  Response. This value is not returned when `DCRPC_NOREPLY` is specified in *data-name-C*.

(5) Status codes

Status code	Meaning
00000	Normal termination
02401	Invalid value for the data name. The request code (*data-name-A*) may be invalid.
02402	`CBLDCRPS('OPEN    ')` has not been executed.
02403	One of the following errors occurs. • Initialization failed. • No user authentication is performed. • The client environment definition is specified invalidly.
02404	Insufficient memory
02406	Network error
02407	Timeout occurred during execution of `CBLDCRPS('CALL    ')`. Alternatively, the SPP requested to provide a service terminated abnormally before completing the processing.
02408	The input parameter length exceeds the maximum value.
02409	The returned response length exceeds the area provided by the CUP.

Status code	Meaning
02410	The service group name specified for *data-name-F* is undefined.
02411	The service name specified for *data-name-E* is undefined.
02412	The service group that contains the service specified with *data-name-E* is shut down.
02413	The specified service is being terminated.
02414	The SPP requested to provide a service was not started, or terminated abnormally before completing the processing. This value is returned when 0 is specified for `DCWATCHTIM` in the client environment definition (infinite response wait is specified).
02415	OpenTP1 is inactive for the node corresponding to the specified service.
02416	The specified service caused a system error.
02417	The specified service caused insufficient memory.
02418	System error
02419	The response length returned from the service program to OpenTP1 is outside the range from 1 to `DCRPC_MAX_MESSAGE_SIZE`#.
02420	OpenTP1 is starting up on the service-requested node.
02423	Insufficient memory
02424	System error
02425	The specified service caused a system error.
02426	The returned response exceeds the area provided by the CUP.
02427	Two or more SPPs use different transaction attributes when the inter-node load balancing facility is used. This status code returns only when a service is requested for SPP that uses the inter-node load balancing facility.
02442	A permanent connection was released.
02456	A service request was issued to the socket-receiving type server, which could not receive the request.

Status code	Meaning
02466	A service request was issued to an SPP for which `test_mode=no` was specified in the user service definition in an environment where `DCUTOKEY` was specified in the client environment definition. Alternatively, a function was called in an environment where the following conditions were satisfied: • `DCUTOKEY` was specified in the client environment definition. • A permanent connection with the CUP executing process was being established. • The service request was issued outside the transaction. • A service request was issued to an SPP for which a value other than `test_mode=no` was specified in the user service definition.
02467	After a chained RPC has been used for transaction processing, `CBLDCRPS('CALL    ')` that sets 32 as data name C issues a service request.
02470	The service-requested SPP is protected by the security facility. The UAP that called `CBLDCRPS('CALL    ')` has no access right to the server UAP.
02472	Transaction branches cannot be started because the number of transaction branches that can be started concurrently has been exceeded, or because the maximum number of child transaction branches that can be started from one transaction branch has been exceeded. Alternatively, 32 is not set at data name C in a service request qualified by a domain in a transaction.
02478	The SPP requested to provide a service terminated abnormally before completing the processing. This value is returned when `00000001` is specified for `DCEXTENDFUNCTION` in the client environment definition. If `00000000` is specified or the specification is omitted, 02407 or 02414 returns as the status code.
02479	Since the version of service-requested TP1/Server Base is old (before 03-03), the data compression cannot be used. This status code is returned when the service is requested within the range of the transaction.
02544	The client ID specified for *data-name-G* differs from the one received with `CBLDCCLS('CLTIN   ')` or `CBLDCCLS('EXCLTIN ')`.
02547	The specified port number is in use. Alternatively, port numbers that can be assigned automatically by the operating system are insufficient.

# If you specify 2 or a larger value for `DCCLTRPCMAXMSGSIZE` in the client environment definition, the value you specify is used rather than the value of `DCRPC_MAX_MESSAGE_SIZE` (1 megabyte).

(6) Notes

- Do not specify the same buffer for input parameters and a service program response.
- When *data-name-C* = 1, the following status codes do not return.

  Errors that do not occur

02409

02419

Errors that cannot be detected if occurred

02411

02412

02413

02416

02417

02420

- Status code 02407 may return due to the following conditions.
    - Too small a value is specified as the maximum response wait time in the client environment definition.
    - The service-requested SPP issued a service program, which terminated abnormally.
    - An error occurred on the node that contains the service-requested SPP.
    - The service-requested SSP abnormally terminated before processing of it finishes.
    - A network error occurred.

  Any of these situations may commit the transaction initiated from the service-requested SPP and update the database. Check to see if the database is updated.
- After the CUP executes `CBLDCTRS('BEGIN ')`, executing `CBLDCRPS('CALL ')` may return the following status codes. If so, execute a program that requests rollback as needed.

  02407

  02411

  02417

  02419

  02423

  02424

  02425

  02426

6.3.4 CBLDCRPS('SETWATCH') - Service response wait time update

(1) Form

(a) In a multi-thread environment

■ **PROCEDURE DIVISION**

```
CALL 'CBLDCRPS' USING identifier-1 identifier-2 identifier-3
```

■ **DATA DIVISION**

```
01 identifier-1.
   02 data-name-A PIC X(8)  VALUE 'SETWATCH'.
   02 data-name-B PIC X(5).
   02 FILLER     PIC X(3).
   02 data-name-C PIC S9(9) COMP VALUE ZERO.
   02 data-name-D PIC 9(9)  COMP.
01 identifier-2.
   02 FILLER     PIC X(1).
01 identifier-3.
   02 FILLER     PIC X(1).
```

(b) In a single-thread environment

■ **PROCEDURE DIVISION**

```
CALL 'CBLDCRPC' USING identifier-1
```

■ **DATA DIVISION**

```
01 identifier-1.
   02 data-name-A PIC X(8)  VALUE 'SETWATCH'.
   02 data-name-B PIC X(5).
   02 FILLER     PIC X(3).
   02 data-name-C PIC S9(9) COMP VALUE ZERO.
```

(2) Purpose

Changes the timeout for the response of the service request. When the timeout is changed by using this program, the subsequent `CBLDCRPS('CALL    ')` programs will use the new timeout until `CBLDCRPS('CLOSE   ')` is executed. Note that this program does not change the value of `DCWATCHTIM` in the client environment definition.

Before you change the timeout by executing `CBLDCRPS('SETWATCH')`, execute `CBLDCRPS('GETWATCH')` to acquire the current value so that you can restore the previous setting after changing the timeout.

(3) Data area where the UAP sets values

- *data-name-A*

  Set `VALUE 'SETWATCH'` as a request code for updating the service response wait time.

- *data-name-C*

  Set a new service response wait time between 1 and 65535. Specifying `0` means an infinite wait.

- *data-name-D*

  Specify the client ID received with `CBLDCCLS('CLTIN    ')` or `CBLDCCLS('EXCLTIN ')`.

(4) Data area for which a value is returned

- *data-name-B*

  5-digit status code.

(5) Status codes

Status code	Meaning
00000	Normal termination
02401	Invalid value for the data name. The request code (*data-name-A*) may be invalid.
02402	`CBLDCRPS('OPEN    ')` is not executed.
02404	Insufficient memory
02544	The client ID specified for *data-name-D* differs from the one received with `CBLDCCLS('CLTIN    ')` or `CBLDCCLS('EXCLTIN ')`.

6.3.5 CBLDCRPS('GETWATCH') - Service response wait time reference

(1) Form

(a) In a multi-thread environment

- **`PROCEDURE DIVISION`**

`CALL 'CBLDCRPS' USING` *identifier-1 identifier-2 identifier-3*

■ **DATA DIVISION**

```
01 identifier-1.
   02 data-name-A PIC X(8)  VALUE 'GETWATCH'.
   02 data-name-B PIC X(5).
   02 FILLER     PIC X(3).
   02 data-name-C PIC S9(9) COMP VALUE ZERO.
   02 data-name-D PIC 9(9)  COMP.
01 identifier-2.
   02 FILLER     PIC X(1).
01 identifier-3.
   02 FILLER     PIC X(1).
```

(b) In a single-thread environment

■ **PROCEDURE DIVISION**

```
CALL 'CBLDCRPC' USING identifier-1
```

■ **DATA DIVISION**

```
01 identifier-1.
   02 data-name-A PIC X(8)  VALUE 'GETWATCH'.
   02 data-name-B PIC X(5).
   02 FILLER     PIC X(3).
   02 data-name-C PIC S9(9) COMP VALUE ZERO.
```

(2) Purpose

References the response wait time for the current service request.

You can use `CBLDCRPS('GETWATCH')` to acquire the current timeout for the response of the service request so that you can restore the previous setting after temporarily changing the timeout by using `CBLDCRPS('SETWATCH')`.

The `CBLDCRPS('GETWATCH')` program returns the service response time that is changed by `CBLDCRPS('SETWATCH')`. If the service response time is unchanged, the program returns the `DCWATCHTIM` value in the client environment definition.

Returned values are available for `CBLDCRPS('CALL    ')` for OpenTP1.

(3) Data area where the UAP sets values

■ *data-name-A*

Set `VALUE 'GETWATCH'` as a request code for referencing the service response wait time.

- *data-name-C*

  Set `0`.

- *data-name-D*

  Specify the client ID received with `CBLDCCLS('CLTIN    ')` or `CBLDCCLS('EXCLTIN ')`.

(4) Data area for which a value is returned

- *data-name-B*

  5-digit status code.

- *data-name-C*

  The current value for the maximum time that the system waits for a service response is returned. If 0 is returned, the system waits for a service response indefinitely.

(5) Status codes

Status code	Meaning
00000	Normal termination
02401	The request code (*data-name-A*) may be invalid.
02402	`CBLDCRPS('OPEN     ')` is not executed.
02404	Insufficient memory
02544	The client ID specified for *data-name-D* differs from the one received with `CBLDCCLS('CLTIN    ')` or `CBLDCCLS('EXCLTIN ')`.

6.4 Permanent connection

6.4.1 CBLDCCLS ('CONNECT ') - Establish permanent connection

(1) Form

(a) In a multi-thread environment

■ **PROCEDURE DIVISION**

```
CALL 'CBLDCCLS' USING identifier-1
```

■ **DATA DIVISION**

```
01 identifier-1.
   02 data-name-A PIC X(8)  VALUE 'CONNECT '.
   02 data-name-B PIC X(5).
   02 FILLER     PIC X(3).
   02 data-name-C PIC S9(9) COMP VALUE ZERO.
   02 data-name-D PIC S9(9) COMP.
```

(b) In a single-thread environment

■ **PROCEDURE DIVISION**

```
CALL 'CBLDCCLT' USING identifier-1
```

■ **DATA DIVISION**

```
01 identifier-1.
   02 data-name-A PIC X(8)  VALUE 'CONNECT '.
   02 data-name-B PIC X(5).
   02 FILLER     PIC X(3).
   02 data-name-C PIC S9(9) COMP VALUE ZERO.
```

(2) Purpose

Establishes permanent connection with a CUP execution process, a RAP-processing server or the DCCM3 logical terminal.

The CUP execution process for establishing the permanent connection is running on the OpenTP1 node specified in the *data-name-D* in the `CBLDCCLS('CLTIN    ')`, or specified in `DCCLTRAPHOST` or `DCHOST` in the client environment definition.

To establish the permanent connection with the DCCM3 logical terminal, define `DCCLTDCCMHOST` and `DCCLTDCCMPORT` in the client environment definition. Also specify `32` for *data-name-C* in `CBLDCCLS('CONNECT ')`.

To establish permanent connection with the DCCM3 logical terminal using the remote API facility, provide `DCCLTRAPHOST` with the host name and the port number for the DCCM3 logical terminal. Also specify `0` for *data-name-C* in `CBLDCCLS('CONNECT ')`.

(3) Data area where the UAP sets values

- *data-name-A*

  Set `VALUE 'CONNECT△'` as the request code for indicating establishment of permanent connection.

- *data-name-C*

  Specify the node with which you want to establish permanent connection.

  `0`: Permanent connection is established with TP1/Server, a RAP-processing server or the DCCM3 logical terminal.

  `32`: Permanent connection is established with the DCCM3 logical terminal.

- *data-name-D*

  Specify the client ID received with `CBLDCCLS('CLTIN   ')` or `CBLDCCLS('EXCLTIN ')`.

(4) Data area for which a value is returned

- *data-name-B*

  5-digit status code.

(5) Status codes

Status code	Meaning
00000	Normal termination. Or, permanent connection has already been established.
02501	Invalid value for the data name. The request code (*data-name-A*) may be invalid.
02502	• `CBLDCCLS('CONNECT ')` is issued in the transaction, or `CBLDCRPS ('OPEN ')` is not issued. • The establishment request to OpenTP1 is issued while permanent connection with DCCM3 has already been established. • Alternatively, the establishment request to DCCM3 is issued while permanent connection with OpenTP1 has already been established.
02504	A necessary amount of buffer could not be allocated.
02506	Communication error

Status code	Meaning
02507	A timeout error occurred during establishment of permanent connection.
02515	One of the following causes is likely: • The OpenTP1 server or the DCCM3 logical terminal has not started. • The client extended service has not started. Check whether `clt_conf` is specified correctly in the system service configuration definition. • The CUP executing process has not started. Check whether `clt_cup_conf` is specified correctly in the client service definition.
02518	System error
02539	The establishment request to the DCCM3 logical terminal is issued with an invalid host name.
02544	The client ID specified in *data-name-D* differs from the one received with `CBLDCCLS('CLTIN ')` or `CBLDCCLS('EXCLTIN ')`.
02547	The specified port number is in use, or port numbers that can be assigned automatically by the operating system are insufficient.

(6) Notes

- No permanent connection is established when `CBLDCCLS('CONNECT ')` returns error. Permanent connection may be established only on the CUP execution process if the error is returned with the status code 02506, 02507, or 02518.

  In this case, the CUP execution process or DCCM3 logical terminal may keep on waiting for a response from the CUP. To prevent an infinite wait, specify an appropriate value for the maximum time interval for the permanent connection. For a DCCM3 logical terminal, specify an appropriate value for the time during which the system is unable to determine whether a connection with the terminal is valid.
- `CBLDCCLS('CONNECT ')` cannot be issued in a transaction.
- You can establish permanent connection with only one of the following two categories.
- CUP execution process, RAP-processing server, or a DCCM3 logical terminal that is specified for `DCCLTRAPHOST` in the client environment definition
- DCCM3 logical terminal that is specified for `DCCLTDCCMHOST` in the client environment definition

  If you establish permanent connection with one category, you cannot communicate with the other until you issue `CBLDCCLS('DISCNCT ')`.
- The data compression is unavailable when you establish permanent connection with DCCM3 logical terminals. You need to omit `DCCLTDATACOMP` or specify `N` for it in the client environment definition.

6.4.2 CBLDCCLS ('DISCNCT ') - Release permanent connection

(1) Form

(a) In a multi-thread environment

■ **PROCEDURE DIVISION**

```
CALL 'CBLDCCLS' USING identifier-1
```

■ **DATA DIVISION**

```
01 identifier-1.
   02 data-name-A PIC X(8)  VALUE 'DISCNCT '.
   02 data-name-B PIC X(5).
   02 FILLER     PIC X(3).
   02 data-name-C PIC S9(9) COMP VALUE ZERO.
   02 data-name-D PIC 9(9) COMP.
```

(b) In a single-thread environment

■ **PROCEDURE DIVISION**

```
CALL 'CBLDCCLT' USING identifier-1
```

■ **DATA DIVISION**

```
01 identifier-1.
   02 data-name-A PIC X(8)  VALUE 'DISCNCT '.
   02 data-name-B PIC X(5).
   02 FILLER     PIC X(3).
   02 data-name-C PIC S9(9) COMP VALUE ZERO.
```

(2) Purpose

Releases the permanent connection with a CUP execution process, a RAP-processing server or the DCCM3 logical terminal.

(3) Data area where the UAP sets values

- *data-name-A*

  Set `VALUE 'DISCNCT△'` as the request code for indicating release of permanent connection.

- *data-name-C*

Set `0`.

- *data-name-D*

  Specify the client ID received with `CBLDCCLS('CLTIN    ')` or `CBLDCCLS('EXCLTIN ')`.

(4) Data area for which a value is returned

- *data-name-B*

  5-digit status code.

(5) Status codes

Status code	Meaning
00000	Normal termination. Alternatively, for TP1/Client/W, the permanent connection is already disconnected.
02501	Invalid value for the data name. The request code (*data-name-A*) may be invalid.
02502	`CBLDCCLS('DISCNCT ')` is issued in the transaction, or `CBLDCCLS('OPEN     ')` is not issued.
02504	A necessary amount of buffer could not be allocated.
02506	Communication error. Alternatively, for TP1/Client/P, the permanent connection is already disconnected.
02507	A timeout error occurred during establishment of permanent connection.
02518	System error
02544	The client ID specified in *data-name-D* differs from the one received with `CBLDCCLS('CLTIN    ')` or `CBLDCCLS('EXCLTIN ')`.

(6) Notes

- The permanent connection is not released if `CBLDCCLS('DISCNCT ')` returns an error with either of the following status codes:
  - 02501
  - 02502
  - 02504 (when the error is detected on the client)
  - 02544
- When `CBLDCCLS('DISCNCT ')` returns an error with one of the following status codes, TP1/Client forcibly releases the permanent connection.
  - 02504 (when the error is detected on the server)

- 02506
- 02507
- 02518

In this case, the CUP execution process or DCCM3 logical terminal may keep on waiting for a response from the CUP, without detecting the release of permanent connection by TP1/Client. To prevent an infinite wait, specify an appropriate value for the maximum time interval for the permanent connection. For a DCCM3 logical terminal, specify an appropriate value for the time during which the system is unable to determine whether a connection with the terminal is valid.

Issuing `CBLDCCLS('DISCNCT ')` in a transaction commits the transaction.

6.4.3 CBLDCCLS('STRAPHST') - Set the destination of a request to establish a permanent connection

(1) Form

(a) In a multi-thread environment

■ **PROCEDURE DIVISION**

```
CALL 'CBLDCCLS' USING identifier-1
```

■ **DATA DIVISION**

```
01 identifier-1.
      02  data-name-A   PIC X(8) VALUE 'STRAPHST'.
      02  data-name-B   PIC X(5).
      02  FILLER        PIC X(3).
      02  data-name-C   PIC S9(9) COMP VALUE ZERO.
      02  data-name-D   PIC 9(9) COMP.
      02  data-name-E   PIC X(n).
```

(b) In a single-thread environment

■ **PROCEDURE DIVISION**

```
CALL 'CBLDCCLT' USING identifier-1
```

■ **DATA DIVISION**

```
01 identifier-1.
      02  data-name-A   PIC X(8) VALUE 'STRAPHST'.
```

```
02  data-name-B   PIC X(5).
02  FILLER         PIC X(3).
02  data-name-C   PIC S9(9) COMP VALUE ZERO.
02  FILLER         PIC 9(9) COMP.
02  data-name-E   PIC X(n).
```

(2) Purpose

CBLDCCLS('STRAPHST') sets the host name and port number of the node to which you want to send a request to establish a permanent connection. The host name and port number set by these programs prevail over those specified in DCCLTRAPHOST in the client environment definition. After CBLDCCLS('STRAPHST') is executed, CBLDCCLS('CONNECT ') uses the host name and port number specified in CBLDCCLS('STRAPHST').

You may want to restore the host name and port number that were used before CBLDCCLS('STRAPHST') was executed. To do this, before executing CBLDCCLS('STRAPHST') to set a new host name and port number, execute CBLDCCLS('GTRAPHST') to acquire the current host name and port number. Then, after executing CBLDCCLS('STRAPHST') to set a new host name and port number, reexecute the function specifying the previously acquired host name and port number.

(3) Data area where the UAP sets values

- *data-name-A*

  Set VALUE 'STRAPHST' as a request code for setting the destination of a request to establish a permanent connection.

- *data-name-C*

  Set 0.

- *data-name-D*

  Specify the client ID received with CBLDCCLS('CLTIN   ') or CBLDCCLS('EXCLTIN ').

- *data-name-E*

  Specify the host name and port number of the node to which you want to send a request to establish a permanent connection. You can specify an IP address in decimal dot notation for the host name.

  Form:

  host-name[:*port-number*][,*host-name*[:*port-number*],...]

  - *host-name* ~<character string>

    In *host-name*, specify the host name of the node to which you want to send a request to establish a permanent connection.

You can specify a maximum of 63# characters for the host name. When specifying multiple host names, you can specify a maximum of 255# characters, including port numbers, in *data-name-E*.

- *port-number* ~<unsigned integer>((5001 to 65535))

  In *port-number*, specify the port number of the node to which you want to send a request to establish a permanent connection.

  # If you specify `00000008` for `DCCLTOPTION` in the client environment definition, you can specify a maximum of 255 characters for the host name. When specifying multiple host names, you can specify a maximum of 1023 characters, including port numbers, in *data-name-E*.

(4) Data area for which a value is returned

■ *data-name-B*

A five-digit status code is returned.

(5) Status codes

Status code	Meaning
00000	The program normally terminated.
02501	The value specified in an argument is incorrect. The request code (*data-name-A*) may be incorrect.
02502	Possible causes are as follows: • The program has already been executed in the transaction. • A permanent connection is being established. • `CBLDCRPS('OPEN    ')` has not been executed.
02504	A necessary amount of buffer could not be allocated.
02544	The client ID specified in *data-name-D* differs from the one received with `CBLDCCLS('CLTIN   ')` or `CBLDCCLS('EXCLTIN ')`.

(6) Notes

- This function does not change the value specified in `DCCLTRAPHOST` in the client environment definition.
- If you specify a blank at the beginning of *data-name-E*, `DCCLTRAPHOST` is placed in undefined status in the client environment definition. When `DCCLTRAPHOST` is not defined, `CBLDCCLS('CONNECT ')` establishes a permanent connection to the logical terminal of the CUP executing process or of DCCM3.

6.4.4 CBLDCCLS('GTRAPHST') - Acquire the destination of a request to establish a permanent connection

(1) Form

(a) In a multi-thread environment

■ **PROCEDURE DIVISION**

```
CALL 'CBLDCCLS' USING identifier-1
```

■ **DATA DIVISION**

```
01 identifier-1.
      02  data-name-A  PIC X(8) VALUE 'GTRAPHST'.
      02  data-name-B  PIC X(5).
      02  FILLER       PIC X(3).
      02  data-name-C  PIC S9(9) COMP VALUE ZERO.
      02  data-name-D  PIC 9(9) COMP.
      02  data-name-F  PIC X(n).
```

(b) In a single-thread environment

■ **PROCEDURE DIVISION**

```
CALL 'CBLDCCLT' USING identifier-1
```

■ **DATA DIVISION**

```
01 identifier-1.
      02  data-name-A  PIC X(8) VALUE 'GTRAPHST'.
      02  data-name-B  PIC X(5).
      02  FILLER       PIC X(3).
      02  data-name-C  PIC S9(9) COMP VALUE ZERO.
      02  data-name-D  PIC 9(9) COMP.
      02  data-name-E  PIC X(n).
```

(2) Purpose

`CBLDCCLS('GTRAPHST')` acquires the host name and port number of the node to which you want to send a request to establish a permanent connection.

Before executing `CBLDCCLS('STRAPHST')` to specify the new destination of a request to establish a permanent connection, execute `CBLDCCLS('GTRAPHST')` to save the current destination.

When `CBLDCCLS('GTRAPHST')` is executed, the latest destination set by `CBLDCCLS('STRAPHST')` is returned to *data-name-E*. If `CBLDCCLS('STRAPHST')` has not been executed, the value of `DCCLTRAPHOST` in the client environment definition is returned to *data-name-E*.

(3) Data area where the UAP sets values

- *data-name-A*

  Set `VALUE 'GTRAPHST'` as a request code for acquiring the destination of a request to establish a permanent connection.

- *data-name-C*

  Set `0`.

- *data-name-D*

  Specify the client ID received with `CBLDCCLS('CLTIN    ')` or `CBLDCCLS('EXCLTIN ')`.

- *data-name-E*

  Specify an area of 256 bytes# or larger for storing the host name and port number that are currently set as the destination of a request for establishing a permanent connection.

  # If you specify `00000008` for `DCCLTOPTION` in the client environment definition, this value is 1,024 bytes, not 256 bytes.

(4) Data area for which a value is returned

- *data-name-B*

  A five-digit status code is returned.

- *data-name-E*

  The currently set host name and port number of the node that is currently set as the destination of a request to establish a permanent connection is returned. If the destination is not specified by `CBLDCCLS('STRAPHST')` when `DCCLTRAPHOST` is not specified in the client environment definition, a space is returned at the beginning of *data-name-E*.

  Form:

  host-name[:*port-number*][,*host-name*[:*port-number*],...]

  host-name ~<character string>

  The host name of the destination of a request to establish a permanent connection is returned.

port-number ~<unsigned integer>((5001 to 65535))

The port number of the destination of a request to establish a permanent connection is returned.

(5) Status codes

Status code	Meaning
00000	The program normally terminated.
02501	The value specified in an argument is incorrect. The request code (*data-name-A*) may be incorrect.
02502	`CBLDCRPS('OPEN     ')` has not been executed.
02504	A necessary amount of buffer could not be allocated.
02544	The client ID specified in *data-name-D* differs from the one acquired by `CBLDCCLS('CLTIN   ')` or `CBLDCCLS('EXCLTIN ')`.

6.4.5 CBLDCCLS('STCONINF') - Set terminal identification information

(1) Form

(a) In a multi-thread environment

■ **PROCEDURE DIVISION**

```
CALL  'CBLDCCLS'  USING  identifier-1
```

■ **DATA DIVISION**

```
01 identifier-1.
      02  data-name-A  PIC X(8)  VALUE 'STCONINF'.
      02  data-name-B  PIC X(5).
      02  FILLER  PIC X(3).
      02  data-name-C  PIC S9(9) COMP VALUE ZERO.
      02  data-name-D  PIC 9(4)  COMP.
      02  FILLER  PIC X(2).
      02  data-name-F  PIC 9(9)  COMP.
      02  data-name-G  PIC X(n).
```

(b) In a single-thread environment

■ **PROCEDURE DIVISION**

```
CALL  'CBLDCCLT'  USING  identifier-1
```

■ **DATA DIVISION**

```
01 identifier-1.
      02  data-name-A  PIC X(8)  VALUE 'STCONINF'.
      02  data-name-B  PIC X(5).
      02  FILLER  PIC X(3).
      02  data-name-C  PIC S9(9) COMP VALUE ZERO.
      02  data-name-D  PIC 9(4) COMP.
      02  FILLER  PIC X(2).
      02  data-name-E  PIC 9(9) COMP.
      02  data-name-G  PIC X(n).
```

(2) Purpose

`CBLDCCLS('STCONINF')` dynamically sets terminal identification information.

When TP1/Client communicates with a DCCM3 logical terminal over a permanent connection, the DCCM3 function for allocating a fixed terminal can be used provided terminal identification information is reported to the DCCM3 logical terminal.

The terminal identification information is set in *data-name-G* of this request code. However, the setting takes effect only when the host name and port number of the DCCM3 logical terminal are specified for `DCCLTRAPHOST` in the client environment definition, and `0` is specified in *data-name-C* of the `CBLDCCLS('CONNECT ')` statement. The `CBLDCCLS('CONNECT ')` statement executed after this request code references the terminal identification information, and reports the information to the DCCM3 logical terminal.

When this request code is executed, the terminal identification information specified for `DCCLTCONNECTINF` in the client environment definition is not referenced until `CBLDCRPS('OPEN    ')` is executed again.

If this request code is executed more than once, the terminal identification information specified immediately before execution of `CBLDCCLS('CONNECT ')` takes effect.

(3) Data area where the UAP sets values

- *data-name-A*

  Set `VALUE 'STCONINF'` as a request code for setting terminal identification information.

- *data-name-C*

  Set `0`.

- *data-name-D*

  Set the length of the terminal identification information.

- *data-name-E*

  This area is used by OpenTP1.

- *data-name-F*

  Set the client ID received by `CBLDCCLS('CLTIN    ')` or `CBLDCCLS('EXCLTIN ')`.

- *data-name-G*

  Set terminal identification information.

(4) Data area for which a value is returned

- *data-name-B*

  5-digit status code

(5) Status codes

Status code	Meaning
00000	Normal termination
02501	Invalid value for the data name. The request code (*data-name-A*) may be invalid.
02502	`CBLDCRPS('OPEN     ')` has not been issued.
02504	A necessary amount of buffer could not be allocated.
02544	The client ID specified in data-name-F is different from the client ID received by `CBLDCCLS('CLTIN    ')` or `CBLDCCLS('EXCLTIN ')`.

(6) Notes

- Only if the DCCM3 version is 09-03 or later, can the DCCM3 function for allocating a fixed terminal be used by reporting terminal identification information to the DCCM3 logical terminal. For details about the function for allocating a fixed terminal, see the manual *VOS3 Data Management System XDM E2 Description*.
- If the DCCM3 logical terminal name corresponding to the terminal identification information defined by `CBLDCCLS('STCONINF')` has not been defined in DCCM3, status code 02506 is returned.

6.5 Transaction control

6.5.1 CBLDCTRS('BEGIN   ') - Transaction startup

(1) Form

(a) In a multi-thread environment

■ **PROCEDURE DIVISION**

```
CALL 'CBLDCTRS' USING identifier-1
```

■ **DATA DIVISION**

```
01 identifier-1.
   02 data-name-A PIC X(8)  VALUE 'BEGIN  '.
   02 data-name-B PIC X(5).
   02 FILLER     PIC X(3).
   02 data-name-C PIC S9(9) COMP.
```

(b) In a single-thread environment

■ **PROCEDURE DIVISION**

```
CALL 'CBLDCTRN' USING identifier-1
```

■ **DATA DIVISION**

```
01 identifier-1.
   02 data-name-A PIC X(8)  VALUE 'BEGIN  '.
   02 data-name-B PIC X(5).
```

(2) Purpose

Starts a global transaction from the CUP process that executes `CBLDCTRS('BEGIN   ')`.

Issue `CBLDCTRS('BEGIN   ')` after executing `CBLDCRPS('OPEN    ')`.

One global transaction means a process between the point where `CBLDCTRS('BEGIN   ')` is executed and a synchronization point (request to commit) for the transaction.

You cannot issue `CBLDCTRS('BEGIN   ')` twice or more within the global transaction. This also applies to `CBLDCTRS('BEGIN   ')` for the SPP. If the program is issued against this rule, it returns an error.

The SPP transaction attribute follows the specification of `atomic_update` in the user service definition.

(3) Data area where the UAP sets values

- *data-name-A*

  Set `VALUE 'BEGIN△△△'` as the request code for indicating the start of a transaction.

- *data-name-C*

  Specify the client ID received with `CBLDCCLS('CLTIN    ')` or `CBLDCCLS('EXCLTIN ')`.

(4) Data area for which a value is returned

- *data-name-B*

  5-digit status code.

(5) Status codes

Status code	Meaning
00000	Normal termination
02501	The request code (*data-name-A*) may be invalid.
02502	The program was issued from an incorrect context (for example, the program was issued within a transaction). Alternatively, the function has been issued from an environment where both of the following conditions exist: • `DCUTOKEY` is specified in the client environment definition. • A permanent connection is being established with a RAP-processing server.
02504	Insufficient memory
02506	Network error
02507	Timeout occurred during processing of `CBLDCTRS('BEGIN    ')`.
02510	The client extended service has not started. Check whether `clt_conf` is specified correctly in the system service configuration definition. Alternatively, the transactional RPC executing process has not started. Check whether `clt_trn_conf` is specified correctly in the client service definition.
02515	OpenTP1 is inactive.
02517	An insufficient memory condition occurred within the transaction process.
02518	System error

Status code	Meaning
02542	The permanent connection was released from the CUP executing process.
02544	The client ID specified for *data-name-C* differs from the one received with `CBLDCCLS('CLTIN   ')` or `CBLDCCLS('EXCLTIN ')`.
02545	A new transaction could not be started because the server's transaction processing was overloaded. If this status code returns, it is highly possible that the program will succeed if reexecuted. Reexecute the program.
02547	The specified port number is in use, or port numbers that can be assigned automatically by the operating system are insufficient.
03406	An error occurred in the resource manager (RM). No transaction could occur.
03407	A transaction could not be started because an error occurred in the transaction service. If this status code returns, it is highly possible that the program will succeed if reexecuted. Reexecute the program.

6.5.2 CBLDCTRS('C-COMMIT') - Commit in chained mode

(1) Form

(a) In a multi-thread environment

■ **PROCEDURE DIVISION**

```
CALL 'CBLDCTRS' USING identifier-1
```

■ **DATA DIVISION**

```
01 identifier-1.
   02 data-name-A PIC X(8)  VALUE 'C-COMMIT'.
   02 data-name-B PIC X(5).
   02 FILLER      PIC X(3).
   02 data-name-C PIC 9(9) COMP.
```

(b) In a single-thread environment

■ **PROCEDURE DIVISION**

```
CALL 'CBLDCTRN' USING identifier-1
```

■ **DATA DIVISION**

```
01 identifier-1.
```

```
02 data-name-A PIC X(8)  VALUE 'C-COMMIT'.
02 data-name-B PIC X(5).
```

(2) Purpose

Acquires a synchronous point for the transaction.

When `CBLDCTRS('C-COMMIT')` terminates normally, a new global transaction occurs. It controls succeeding programs.

(3) Data area where the UAP sets values

- *data-name-A*

  Set `VALUE 'C-COMMIT'` as a request code for committing in chained mode. This value is unchanged for processing after the commit statement in chained mode.

- *data-name-C*

  Specify the client ID received with `CBLDCCLS('CLTIN    ')` or `CBLDCCLS('EXCLTIN ')`.

(4) Data area for which a value is returned

- *data-name-B*

  5-digit status code.

(5) Status codes

Status code	Meaning
00000	Normal termination
02501	The request code (*data-name-A*) may be invalid.
02502	The program is issued from an incorrect context.
02504	Insufficient memory
02506	Network error
02507	A timeout error occurred in the processing for `CBLDCTRS('C-COMMIT')`.
02515	OpenTP1 is inactive.
02517	An insufficient memory condition occurred within the transaction process.
02518	System error
02542	The permanent connection has been released.
02544	The client ID specified for *data-name-C* differs from the one received with `CBLDCCLS('CLTIN    ')` or `CBLDCCLS('EXCLTIN ')`.

Status code	Meaning
03402	The current transaction could not be committed and rolled back. The process is now under the transaction and is within the scope of the global transaction.
03403	Due to the heuristic determination, some transaction branches are committed and others are rolled back. This status code will be returned if the result of the heuristic determination differs from the result of the synchronous point for the global transaction. For the cause of this status code or the result of the synchronous point for the global transaction, see the message log file. After this status code returns, the process is still under the transaction and is within the scope of the global transaction.
03404	The global transaction's transaction branch has completed heuristically. But an error makes it impossible to determine the result of the synchronous point for this transaction branch. For the cause of this status code or the result of the synchronous point for the global transaction, see the message log file. After this status code returns, the process is still under the transaction and is within the scope of the global transaction.
03424	The transaction has been committed normally. But new transactions could not start. When this status code returns, the process is no more under control of the transaction.
03425	The current transaction cannot be committed and is rolled back. New transactions could not start. The process is no more under the transaction.
03426	The global transaction that executed the `CBLDCTRS ('C-COMMIT')` function follows the heuristic determination. Some transactions may or may not be committed. This status code will be returned if the result of the heuristic determination differs from the result of the synchronous point for the global transaction. For the result of the synchronous point for the UAP, resource manager, or global transaction that caused this status code, see the message log file. New transactions could not start. The process is not under the transaction.
03427	The global transaction's transaction branch has completed heuristically. But an error makes it impossible to determine the result of the synchronous point for this transaction branch. For the result of the synchronous point for the UAP, resource manager, or global transaction that caused this status code, see the message log file. New transactions could not start. The process is not under the transaction. If this status code returns, the process is still under the transaction and within the scope of the global transaction.

(6) Notes

To terminate a CUP process after committing a transaction, make sure that you execute `CBLDCTRS('U-COMMIT')`.

6.5.3 CBLDCTRS('C-ROLL ') - Rollback in chained mode

(1) Form

(a) In a multi-thread environment

■ **PROCEDURE DIVISION**

```
CALL 'CBLDCTRS' USING identifier-1
```

■ **DATA DIVISION**

```
01 identifier-1.
   02 data-name-A PIC X(8)  VALUE 'C-ROLL  '.
   02 data-name-B PIC X(5).
   02 FILLER     PIC X(3).
   02 data-name-C PIC S9(9) COMP.
```

(b) In a single-thread environment

■ **PROCEDURE DIVISION**

```
CALL 'CBLDCTRN' USING identifier-1
```

■ **DATA DIVISION**

```
01 identifier-1.
   02 data-name-A PIC X(8)  VALUE 'C-ROLL  '.
   02 data-name-B PIC X(5).
```

(2) Purpose

Rolls back the transaction.

When `CBLDCTRS('C-ROLL  ')` terminates normally, a new global transaction occurs. It controls succeeding programs.

(3) Data area where the UAP sets values

- *data-name-A*

  Set `VALUE 'C-ROLL△ △'` as a request code for rolling back in chained mode.

- *data-name-C*

  Specify the client ID received with `CBLDCCLS('CLTIN   ')` or `CBLDCCLS('EXCLTIN ')`.

(4) Data area for which a value is returned

- *data-name-B*

  5-digit status code.

(5) Status codes

Status code	Meaning
00000	Normal termination
02501	The request code (*data-name-A*) may be invalid.
02502	The program is issued from an incorrect context.
02504	Insufficient memory
02506	Network error
02507	Timeout occurred during processing of `CBLDCTRS('C-ROLL   ')`.
02515	OpenTP1 is inactive.
02517	An insufficient memory condition occurred within the transaction process.
02518	System error
02542	The permanent connection has been released.
02544	The client ID specified for *data-name-C* differs from the one received with `CBLDCCLS('CLTIN    ')` or `CBLDCCLS('EXCLTIN ')`.
03403	Due to the heuristic determination, some transaction branches are committed and others are rolled back. This status code will be returned if the result of the heuristic determination differs from the result of the synchronous point for the global transaction. For the cause of this status code or the result of the synchronous point for the global transaction, see the message log file. After this status code returns, the process is still under the transaction and is within the scope of the global transaction.
03404	The global transaction's transaction branch has completed heuristically. But an error makes it impossible to determine the result of the synchronous point for this transaction branch. For the cause of this status code or the result of the synchronous point for the global transaction, see the message log file. After this status code returns, the process is still under the transaction and is within the scope of the global transaction.
03424	A rollback terminated normally but a new transaction could not be started. When this status code returns, the process is no more under control of the transaction.

Status code	Meaning
03426	The global transaction that executed the `CBLDCTRS ('C-COMMIT)` function follows the heuristic determination. Some transactions may or may not be committed. This status code will be returned if the result of the heuristic determination differs from the result of the synchronous point for the global transaction. For the result of the synchronous point for the UAP, resource manager, or global transaction that caused this status code, see the message log file. New transactions could not start. The process is not under the transaction.
03427	The global transaction's transaction branch has completed heuristically. But an error makes it impossible to determine the result of the synchronous point for this transaction branch. For the result of the synchronous point for the UAP, resource manager, or global transaction that caused this status code, see the message log file. New transactions could not start. The process is not under the transaction. If this status code returns, the process is still under the transaction and within the scope of the global transaction.

(6) Notes

When terminating the CUP process by rolling back the transaction, be sure to execute `CBLDCTRS('U-ROLL  ')`.

6.5.4 CBLDCTRS('U-COMMIT') - Commit in unchained mode

(1) Form

(a) In a multi-thread environment

■ **PROCEDURE DIVISION**

```
CALL 'CBLDCTRS' USING identifier-1
```

■ **DATA DIVISION**

```
01 identifier-1.
   02 data-name-A PIC X(8)  VALUE 'U-COMMIT'.
   02 data-name-B PIC X(5).
   02 FILLER     PIC X(3).
   02 data-name-C PIC 9(9) COMP.
```

(b) In a single-thread environment

■ **PROCEDURE DIVISION**

```
CALL 'CBLDCTRN' USING identifier-1
```

■ **DATA DIVISION**

```
01 identifier-1.
   02 data-name-A PIC X(8)  VALUE 'U-COMMIT'.
   02 data-name-B PIC X(5).
```

(2) Purpose

Acquires the transaction's synchronous point.

When `CBLDCTRS('U-COMMIT')` terminates normally, the global transaction terminates. Outside the scope of the global transaction, the SPP cannot execute as a transaction.

(3) Data area where the UAP sets values

- *data-name-A*

  Set `VALUE 'U-COMMIT'` as a request code for indicating the commit in unchained mode. The content of this area is unchanged for processing after the commit statement in unchained mode.

- *data-name-C*

  Specify the client ID received with `CBLDCCLS('CLTIN   ')` or `CBLDCCLS('EXCLTIN ')`.

(4) Data area for which a value is returned

- *data-name-B*

  5-digit status code.

(5) Status codes

Status code	Meaning
00000	Normal termination
02501	The request code (*data-name-A*) may be invalid.
02502	The program is issued from an incorrect context.
02504	Insufficient memory
02506	Network error
02507	A timeout error occurred in the processing for `CBLDCTRS('U-COMMIT')`.
02515	OpenTP1 is inactive.
02517	An insufficient memory condition occurred within the transaction process.

Status code	Meaning
02518	System error
02542	The permanent connection has been released.
02544	The client ID specified for *data-name-C* differs from the one received with `CBLDCCLS('CLTIN   ')` or `CBLDCCLS('EXCLTIN ')`.
03402	The transaction was rolled back because a commit failed. After this status code returns, the process is outside the scope of the global transaction.
03403	Due to the heuristic determination, some or all transaction branches rolled back. See the message log file for details. After this status code returns, the process is outside the scope of the global transaction.
03404	The heuristic determination completed the transaction. But an error makes it impossible to determine the result. See the message log file for details. After this status code returns, the process is outside the scope of the global transaction.

(6) Notes

To normally terminate a CUP process, be sure to execute `CBLDCTRS('U-COMMIT')` to commit the transaction.

6.5.5 CBLDCTRS('U-ROLL ') - Rollback in unchained mode

(1) Form

(a) In a multi-thread environment

■ **PROCEDURE DIVISION**

```
CALL 'CBLDCTRS' USING identifier-1
```

■ **DATA DIVISION**

```
01 identifier-1.
   02 data-name-A PIC X(8)  VALUE 'U-ROLL  '.
   02 data-name-B PIC X(5).
   02 FILLER     PIC X(3).
   02 data-name-C PIC 9(9) COMP.
```

(b) In a single-thread environment

■ **PROCEDURE DIVISION**

```
CALL 'CBLDCTRN' USING identifier-1
```

■ **DATA DIVISION**

```
01 identifier-1.
   02 data-name-A PIC X(8)  VALUE 'U-ROLL  '.
   02 data-name-B PIC X(5).
```

(2) Purpose

When `CBLDCTRS('U-ROLL  ')` terminates normally, the global transaction terminates. Outside the scope of the global transaction, the SPP cannot execute as a transaction.

(3) Data area where the UAP sets values

- *data-name-A*

  Set `VALUE 'U-ROLL△△'` as a request code for indicating the rollback in unchained mode. The content of this area is unchanged for processing after the rollback statement in unchained mode.

- *data-name-C*

  Specify the client ID received with `CBLDCCLS('CLTIN   ')` or `CBLDCCLS('EXCLTIN ')`.

(4) Data area for which a value is returned

- *data-name-B*

  5-digit status code.

(5) Status codes

Status code	Meaning
00000	Normal termination
02501	The request code (*data-name-A*) may be invalid.
02502	The program is issued from an incorrect context.
02504	Insufficient memory
02506	Network error
02507	A timeout error occurred in the processing for `CBLDCTRS('U-ROLL  ')`.
02515	OpenTP1 is inactive.
02517	An insufficient memory condition occurred within the transaction process.
02518	System error

Status code	Meaning
02542	The permanent connection has been released.
02544	The client ID specified for *data-name-C* differs from the one received with `CBLDCCLS('CLTIN   ')` or `CBLDCCLS('EXCLTIN ')`.
03403	Due to the heuristic determination, some or all transaction branches rolled back. See the message log file for details. After this status code returns, the process is outside the scope of the global transaction.
03404	The heuristic determination completed the transaction. But an error makes it impossible to determine the result. See the message log file for details. After this status code returns, the process is outside the scope of the global transaction.

(6) Notes

To normally terminate a CUP process after rolling back the transaction, be sure to execute `CBLDCTRS('U-ROLL  ')` to commit the transaction.

6.5.6 CBLDCTRS('INFO   ') - Post information for current transaction

(1) Form

(a) In a multi-thread environment

■ **PROCEDURE DIVISION**

```
CALL 'CBLDCTRS' USING identifier-1
```

■ **DATA DIVISION**

```
01 identifier-1.
   02 data-name-A   PIC X(8)  VALUE 'INFO    '.
   02 data-name-B   PIC X(5).
   02 FILLER        PIC X(1).
   02 data-name-C
     03 data-name-D PIC S9(4) COMP VALUE ZERO.
   02 data-name-E   PIC 9(9) COMP.
```

(b) In a single-thread environment

■ **PROCEDURE DIVISION**

```
CALL 'CBLDCTRN' USING identifier-1
```

■ **DATA DIVISION**

```
01 identifier-1.
   02 data-name-A   PIC X(8)  VALUE 'INFO    '.
   02 data-name-B   PIC X(5).
   02 FILLER        PIC X(1).
   02 data-name-C
     03 data-name-D   PIC S9(4) COMP VALUE ZERO.
```

(2) Purpose

Checks if the CUP that issued `CBLDCTRS('INFO    ')` is currently working as a transaction.

(3) Data area where the UAP sets values

- *data-name-A*

  Set `VALUE 'INFO△△△△'` as the request code for indicating the report of information about the current transaction.

- *data-name-C*

  Place `0` in this area that stores information about the current transaction.

- *data-name-D*

  Set to `0` the length of an area that stores information about the current transaction. The length should exclude the length of *data-name-D* itself.

- *data-name-E*

  Specify the client ID received with `CBLDCCLS('CLTIN    ')` or `CBLDCCLS('EXCLTIN ')`.

(4) Data area for which a value is returned

- *data-name-B*

  5-digit status code.

(5) Status codes

Status code	Meaning
00001	The CUP process that executed `CBLDCTRS('INFO    ')` is outside the scope of the transaction.
00000	The CUP process that executed `CBLDCTRS('INFO    ')` is within the scope of the transaction.
02501	The request code (*data-name-A*) may be invalid.

Status code	Meaning
02544	The client ID specified for *data-name-E* differs from the one received with `CBLDCCLS('CLTIN   ')` or `CBLDCCLS('EXCLTIN ')`.

6.5.7 CBLDCCLS('GETTRNID') - Collection of identifiers for current transaction

(1) Form

(a) In a multi-thread environment

■ **PROCEDURE DIVISION**

```
CALL 'CBLDCCLS' USING identifier-1
```

■ **DATA DIVISION**

```
01 identifier-1.
   02 data-name-A PIC X(8)  VALUE 'GETTRNID'.
   02 data-name-B PIC X(5).
   02 FILLER     PIC X(3).
   02 data-name-C PIC X(17).
   02 data-name-D PIC X(17).
   02 FILLER     PIC X(2).
   02 data-name-E PIC 9(9) COMP.
```

(b) In a single-thread environment

■ **PROCEDURE DIVISION**

```
CALL 'CBLDCCLT' USING identifier-1
```

■ **DATA DIVISION**

```
01 identifier-1.
   02 data-name-A PIC X(8)  VALUE 'GETTRNID'.
   02 data-name-B PIC X(5).
   02 FILLER     PIC X(3).
   02 data-name-C PIC X(17).
   02 data-name-D PIC X(17).
   02 data-name-E PIC 9(9) COMP.
```

(2) Purpose

Collects the current transaction global identifier and transaction branch identifier.

OpenTP1 allocated these identifiers when any of the following programs started the transaction.

- `CBLDCTRS('BEGIN   ')`
- `CBLDCTRS('C-COMMIT')`
- `CBLDCTRS('C-ROLL  ')`

(3) Data area where the UAP sets values

- *data-name-A*

  Set `VALUE 'GETTRNID'` as a request code for collecting identifiers for the current transaction.

- *data-name-E*

  Specify the client ID received with `CBLDCCLS('CLTIN   ')` or `CBLDCCLS('EXCLTIN ')`.

(4) Data area for which a value is returned

- *data-name-B*

  5-digit status code.

- *data-name-C*

  Transaction global identifier.

- *data-name-D*

  Transaction branch identifier.

(5) Status codes

Status code	Meaning
00000	Normal termination
02501	Invalid value for the data name. The request code (*data-name-A*) may be invalid.
02502	The program is issued from an incorrect context.
02504	Insufficient memory
02544	The client ID specified for *data-name-E* differs from the one received with `CBLDCCLS('CLTIN   ')` or `CBLDCCLS('EXCLTIN ')`.

6.6 TCP/IP communication function

6.6.1 CBLDCCLS('SEND   ') - Sending messages

(1) Form

(a) In a multi-thread environment

■ **PROCEDURE DIVISION**

```
CALL 'CBLDCCLS' USING identifier-1
```

■ **DATA DIVISION**

```
01 identifier-1.
   02 data-name-A  PIC X(8)  VALUE 'SEND   '.
   02 data-name-B  PIC X(5).
   02 FILLER       PIC X(3).
   02 data-name-C PIC S9(9) COMP VALUE ZERO.
   02 data-name-D PIC S9(9) COMP.
   02 data-name-E PIC X(64).
   02 data-name-F PIC 9(4) COMP.
   02 FILLER     PIC X(2).
   02 data-name-H PIC 9(9) COMP.
   02 data-name-I PIC X(n).
```

(b) In a single-thread environment

■ **PROCEDURE DIVISION**

```
CALL 'CBLDCCLT' USING identifier-1
```

■ **DATA DIVISION**

```
01 identifier-1.
   02 data-name-A PIC X(8)  VALUE 'SEND   '.
   02 data-name-B PIC X(5).
   02 FILLER     PIC X(3).
   02 data-name-C PIC S9(9) COMP VALUE ZERO.
   02 data-name-D PIC S9(9) COMP.
   02 data-name-E PIC X(64).
   02 data-name-F PIC 9(4) COMP.
   02 FILLER     PIC X(2).
```

```
  02 data-name-G PIC 9(9) COMP.
  02 data-name-I PIC X(n).
```

(2) Purpose

Sends messages to the MHP.

Before executing `CBLDCCLS('SEND    ')`, execute or `CBLDCRPS('OPEN    ')` by specifying `4` or `16` for *data-name-C*.

(3) Data area where the UAP sets values

- *data-name-A*

  Set `VALUE 'SEND△△△△'` as a request code for sending messages.

- *data-name-C*

  Specify whether to release the connection after sending message.

  `0`: Does not release the connection after sending message.

  `1`: Releases the connection after sending message.

  Except error situations, specifying `0` maintains the connection until you execute `CBLDCRPS('CLOSE   ')`.

- *data-name-D*

  Set the length of a message to be received.

- *data-name-E*

  Specify the host name of the node to be connected when no connection is established. Place a space character at the end of the character string. You can specify an IP address in decimal dot notation for the host name.

  Placing a space character at the beginning of the character string references the content of `DCSNDHOST` in the client environment definition. This content is saved when `CBLDCRPS('OPEN    ')` is executed.

- *data-name-F*

  Specify the port number of the node to be connected by establishing a connection when no connection is established.

  Specifying `0` references the content of `DCSNDPORT` in the client environment definition. This content is saved when `CBLDCRPS('OPEN    ')` is executed.

- *data-name-H*

  Specify the client ID received with `CBLDCCLS('CLTIN   ')` or `CBLDCCLS('EXCLTIN ')`.

- *data-name-I*

  Specify the area for containing the message to be sent. This area must be large enough for the length specified for *data-name-D*.

(4) Data area for which a value is returned

- *data-name-B*

  5-digit status code.

- *data-name-G*

  Area used for OpenTP1.

(5) Status codes

Status code	Meaning
00000	Normal termination
02501	Invalid value for the data name. The request code (*data-name-A*) may be invalid.
02502	Neither `CBLDCRPS('OPEN    ')` is executed. Alternately, it is executed but `4` or `16` is not specified for *data-name-C*.
02504	Insufficient memory
02506	Network error
02507	A request for connection establishment timed out.
02518	System error
02538	Insufficient resource
02539	Incorrect host computer name. Alternatively, no host computer name is specified for *data-name-E* and `DCSNDHOST`.
02541	A request to establish the connection to the remote system was rejected.
02544	The client ID specified for *data-name-H* differs from the one received with `CBLDCCLS('CLTIN   ')` or `CBLDCCLS('EXCLTIN ')`.
02547	Port numbers that can be assigned automatically by the operating system are insufficient.

(6) Note

If the remote system releases the connection when this statement sends a message to the remote system, depending on the length of the message, the statement might not be able to detect that the connection has been released. However, a subsequent request statement might detect it. Keep this in mind when you create a CUP.

6.6.2 CBLDCCLS('EXSEND ') - Sending messages (for an extended host name)

(1) Form

(a) In a multi-thread environment

■ **PROCEDURE DIVISION**

```
CALL 'CBLDCCLS' USING identifier-1  identifier-2  identifier-3
```

■ **DATA DIVISION**

```
01 identifier-1.
   02 data-name-A PIC X(8)  VALUE 'EXSEND  '.
   02 data-name-B PIC X(5).
   02 FILLER     PIC X(3).
   02 data-name-C PIC S9(9) COMP VALUE ZERO.
   02 data-name-D PIC S9(4) COMP.
   02 FILLER     PIC X(2).
   02 data-name-E PIC X(n).
01 identifier-2.
   02 data-name-F PIC S9(9) COMP.
   02 data-name-G PIC X(n).
01 identifier-3.
   02 data-name-H PIC 9(9) COMP.
```

(b) In a single-thread environment

■ **PROCEDURE DIVISION**

```
CALL 'CBLDCCLT' USING identifier-1  identifier-2
```

■ **DATA DIVISION**

```
01 identifier-1.
   02 data-name-A PIC X(8)  VALUE 'EXSEND  '.
   02 data-name-B PIC X(5).
   02 FILLER     PIC X(3).
   02 data-name-C PIC S9(9) COMP VALUE ZERO.
   02 data-name-D PIC 9(4) COMP.
   02 FILLER     PIC X(2).
   02 data-name-E PIC X(n).
01 identifier-2.
```

```
02 data-name-F PIC S9(9) COMP.
02 data-name-G PIC X(n).
```

(2) Purpose

Sends messages to the MHP.

Before executing `CBLDCCLS('EXSEND  ')`, execute `CBLDCRPS('OPEN    ')` by specifying `4` or `16` for *data-name-C*.

Use this function when using the host name extension function.

(3) Data area where the UAP sets values

- *data-name-A*

  Set `VALUE 'EXSEND△△'` as a request code for sending a messages.

- *data-name-C*

  Specify whether to release the connection after sending a message.

  `0`: Does not release the connection after sending message.

  `1`: Releases the connection after sending message.

  Except error situations, specifying `0` maintains the connection until you execute `CBLDCRPS('CLOSE   ')`.

- *data-name-D*

  Specify the port number of the node to be connected by establishing a connection when no connection is established.

  Specifying `0` references the contents of `DCSNDPORT` in the client environment definition which are saved when `CBLDCRPS('OPEN    ')` is executed.

- *data-name-E*

  Specify the host name of the node to be connected when no connection is established. You can specify a maximum of 63# characters for the host name. Terminate the character string with a blank.

  Placing a space character at the beginning of the character string references the contents of `DCSNDHOST` in the client environment definition which are saved when `CBLDCRPS('OPEN    ')` is executed.

  You can specify an IP address in decimal dot notation for the host name.

  # If you specify `00000008` for `DCCLTOPTION` in the client environment definition, you can specify a maximum of 255 characters for the host name.

- *data-name-F*

Set the length of a message to be sent.

- *data-name-G*

  Specify the area for containing the message to be sent. This area must be larger than the length specified for *data-name-F*.

- *data-name-H*

  Specify the client ID received with `CBLDCCLS('CLTIN    ')` or `CBLDCCLS('EXCLTIN ')`.

(4) Data area for which a value is returned

- *data-name-B*

  5-digit status code.

(5) Status codes

Status code	Meaning
00000	Normal termination
02501	Invalid value for the data name. The request code (*data-name-A*) may be invalid.
02502	Possible causes are as follows: `CBLDCRPS('OPEN    ')` has not been executed. `CBLDCRPS('OPEN    ')` has been executed, but `4` or `16` is not specified for *data-name-C*.
02504	Insufficient memory
02506	Network error
02507	A request for connection establishment timed out.
02518	System error
02538	Insufficient resource
02539	The host name is incorrect. Alternatively, no host name is specified for *data-name-E* or `DCSNDHOST`.
02541	A request to establish the connection to the remote system was rejected.
02544	The client ID specified for *data-name-H* differs from the one received with `CBLDCCLS('CLTIN    ')` or `CBLDCCLS('EXCLTIN ')`.
02547	Port numbers that can be assigned automatically by the operating system are insufficient.

(6) Note

If the remote system releases the connection when this statement sends a message to the remote system, depending on the length of the message, the statement might not be able to detect that the connection has been released. However, a subsequent request statement might detect it. Keep this in mind when you create a CUP.

6.6.3 CBLDCCLS('RECEIVE ') - Receiving messages

(1) Form

(a) In a multi-thread environment

■ **PROCEDURE DIVISION**

```
CALL 'CBLDCCLS' USING identifier-1
```

■ **DATA DIVISION**

```
01 identifier-1.
   02 data-name-A PIC X(8)  VALUE 'RECEIVE '.
   02 data-name-B PIC X(5).
   02 FILLER     PIC X(3).
   02 data-name-C PIC S9(9) COMP VALUE ZERO.
   02 data-name-D PIC S9(9) COMP.
   02 data-name-E PIC S9(9) COMP.
   02 data-name-G PIC 9(9) COMP.
   02 data-name-H PIC X(n).
```

(b) In a single-thread environment

■ **PROCEDURE DIVISION**

```
CALL 'CBLDCCLT' USING identifier-1
```

■ **DATA DIVISION**

```
01 identifier-1.
   02 data-name-A PIC X(8)  VALUE 'RECEIVE '.
   02 data-name-B PIC X(5).
   02 FILLER     PIC X(3).
   02 data-name-C PIC S9(9) COMP VALUE ZERO.
   02 data-name-D PIC S9(9) COMP.
   02 data-name-E PIC S9(9) COMP.
   02 data-name-F PIC 9(9) COMP.
   02 data-name-H PIC X(n).
```

(2) Purpose

Receives messages issued from MHP.

Before executing `CBLDCCLS('RECEIVE ')`, execute `CBLDCRPS('OPEN     ')` by specifying `8` or `16` for *data-name-C*.

(3) Data area where the UAP sets values

- *data-name-A*

  Set `VALUE 'RECEIVE△'` as a request code for indicating the rollback in unchained mode.

- *data-name-C*

  Specify whether to release the connection after message reception.

  `0`: Does not release the connection after message reception.

  `2`: Releases the connection after message reception.

  Except error situations, specifying `0` maintains the connection until you issue `CBLDCRPS('CLOSE    ')`.

- *data-name-D*

  Specify the length of a message to be received.

- *data-name-F*

  Area used for OpenTP1.

- *data-name-E*

  Set the maximum time (seconds) for receiving messages between -1 and 65535.

  Specifying `-1` means an infinite wait for a message. Specifying `0` provides no wait for messages. When there are no messages to receive, the program returns an error with status code 02507.

  When you specify a value between 1 and 65535, the program awaits a message for the specified time. When no message is received after the specified time expires, the program returns an error with status code 02507.

- *data-name-G*

  Specify the client ID received with `CBLDCCLS('CLTIN    ')` or `CBLDCCLS('EXCLTIN ')`.

- *data-name-H*

  Area for storing the received message. This area must have at least the length specified for *data-name-D*.

(4) Data area for which a value is returned

- *data-name-B*

  5-digit status code.

- *data-name-H*

  The received message is returned.

(5) Status codes

Status code	Meaning
00000	Normal termination
02501	Invalid value for the data name. The request code (*data-name-A*) may be invalid.
02502	Possible causes are as follows: • `CBLDCRPS('OPEN    ')` has not been executed. • `CBLDCRPS('OPEN    ')` has been executed, but `8` or `16` is not specified for *data-name-C* in `CBLDCRPS('OPEN    ')`.
02504	Insufficient memory
02506	Network error
02507	Timeout occurred during message reception.
02518	System error
02538	Insufficient resource
02542	Connection from the remote system is released.
02544	The client ID specified for *data-name-G* differs from the one received with `CBLDCCLS('CLTIN   ')` or `CBLDCCLS('EXCLTIN ')`.

(6) Notes

- `CBLDCCLS('RECEIVE ')` passes control to the CUP when:
  - The program received a message of the length specified in *data-name-D* from the MHP (control is not returned to the CUP if the received message is shorter than the set length).
  - A timeout occurred while the program was waiting for a message from the MHP.
  - The MHP released the connection.
  - A network error occurred.
- When `CBLDCCLS('RECEIVE ')` is issued, the MHP may release the connection.

In this case, the program returns an error with status code 02542.

6.6.4 CBLDCCLS('RECEIVE2') - Receiving messages (messages receivable even if an error occurs)

(1) Form

(a) In a multi-thread environment

■ **PROCEDURE DIVISION**

```
CALL 'CBLDCCLS' USING identifier-1
```

■ **DATA DIVISION**

```
01 identifier-1.
   02 data-name-A PIC X(8)  VALUE 'RECEIVE2'.
   02 data-name-B PIC X(5).
   02 FILLER     PIC X(3).
   02 data-name-C PIC S9(9) COMP VALUE ZERO.
   02 data-name-D PIC S9(9) COMP.
   02 data-name-E PIC S9(9) COMP.
   02 data-name-G PIC 9(9) COMP.
   02 data-name-H PIC X(n).
```

(b) In a single-thread environment

■ **PROCEDURE DIVISION**

```
CALL 'CBLDCCLT' USING identifier-1
```

■ **DATA DIVISION**

```
01 identifier-1.
   02 data-name-A PIC X(8)  VALUE 'RECEIVE2'.
   02 data-name-B PIC X(5).
   02 FILLER     PIC X(3).
   02 data-name-C PIC S9(9) COMP VALUE ZERO.
   02 data-name-D PIC S9(9) COMP.
   02 data-name-E PIC S9(9) COMP.
   02 data-name-F PIC 9(9) COMP.
   02 data-name-H PIC X(n).
```

(2) Purpose

Receives messages from the MHP.

Before executing `CBLDCCLS('RECEIVE2')`, execute `CBLDCRPS('OPEN    ')` by specifying `8` or `16` for *data-name-C*.

(3) Data area where the UAP sets values

- *data-name-A*

  Set `VALUE 'RECEIVE2'` as a request code for receiving messages.

- *data-name-C*

  Specify whether to release the connection after message reception.

  `0`: Does not release the connection after message reception.

  `2`: Releases the connection after message reception.

  Except error situations, specifying `0` maintains the connection until you issue `CBLDCRPS('CLOSE   ')`.

- *data-name-D*

  Specify the length of the message to receive.

- *data-name-E*

  Set the maximum wait time in seconds for receiving messages. The value must be an integer between -1 and 65535.

  If `-1` is specified:

  The statement waits indefinitely for a message.

  If `0` is specified:

  The statement does not wait for a message. If there is no message to be received, the statement returns a 02507 error.

  If any value from 1 to 65,535 is specified:

  The statement waits for a message, but returns a 02507 error if a message does not arrive within the specified number of seconds.

- *data-name-F*

  An area used for OpenTP1

- *data-name-G*

  Specify the client ID received by `CBLDCCLS('CLTIN   ')` or `CBLDCCLS('EXCLTIN ')`.

- *data-name-H*

An area for storing received messages

(4) Data area for which a value is returned

- *data-name-B*

  5-digit status code

- *data-name-D*

  Length of the received message

- *data-name-H*

  Received message

(5) Status codes

Status code	Meaning
00000	Normal termination
02501	Invalid data name. A request code (*data-name-A*) may be incorrect.
02502	Possible causes are as follows: • `CBLDCRPS('OPEN    ')` has not been executed. • `CBLDCRPS('OPEN    ')` has been executed, but `8` or `16` is not specified for *data-name-C* in `CBLDCRPS('OPEN    ')`.
02504	Insufficient memory
02506	Network error
02507	Timeout occurred during reception of the message.
02518	System error
02538	Insufficient resource
02542	The remote system released the connection.
02544	The client ID specified for *data-name-G* differs from the one received with `CBLDCCLS('CLTIN   ')` or `CBLDCCLS('EXCLTIN ')`.

(6) Notes

- `CBLDCCLS('RECEIVE2')` returns control to the CUP when:
  - The program received a message of the length specified in *data-name-D* from the MHP (control is not returned to the CUP if the received message is shorter than the set length).
  - A timeout occurred while the program was waiting for a message from the MHP.

- The MHP released the connection.
- A network error occurred.

- When `CBLDCCLS('RECEIVE2')` is issued, a disconnection from the MHP allows the program to return an error with 02542.

6.6.5 CBLDCCLS('ASMSEND ') - Send assembled messages

(1) Form

(a) In a multi-thread environment

■ **PROCEDURE DIVISION**

```
CALL 'CBLDCCLS' USING identifier-1  identifier-2  identifier-3
```

■ **DATA DIVISION**

```
01 identifier-1.
   02 data-name-A  PIC X(8)  VALUE 'ASMSEND△'.
   02 data-name-B  PIC X(5).
   02 FILLER    PIC X(3).
   02 data-name-C  PIC S9(9) COMP VALUE ZERO.
   02 data-name-D  PIC 9(4) COMP.
   02 FILLER    PIC X(2).
   02 data-name-E  PIC S9(9) COMP.
   02 FILLER    PIC X(4).
   02 data-name-F  PIC X(n).
01 identifier-2.
   02 data-name-G  PIC S9(9) COMP.
   02 data-name-H  PIC X(n).
01 identifier-3.
   02 data-name-I  PIC 9(9) COMP.
```

(b) In a single-thread environment

■ **PROCEDURE DIVISION**

```
CALL 'CBLDCCLT' USING identifier-1  identifier-2
```

■ **DATA DIVISION**

```
01 identifier-1.
   02 data-name-A  PIC X(8)  VALUE 'ASMSEND△'.
   02 data-name-B  PIC X(5).
```

```
      02 FILLER   PIC X(3).
      02 data-name-C  PIC S9(9) COMP VALUE ZERO.
      02 data-name-D  PIC 9(4) COMP.
      02 FILLER    PIC X(2).
      02 data-name-E  PIC S9(9) COMP.
      02 FILLER    PIC X(4).
      02 data-name-F  PIC X(n).
   01 identifier-2.
      02 data-name-G  PIC S9(9) COMP.
      02 data-name-H  PIC X(n).
```

(2) Purpose

Uses the message assembly facility to send messages. When this facility is used, the statement sends four-byte message information followed by the message body specified in *data-name-H*. If a connection to the remote system has not been established, the function establishes the connection according to the host name specified in *data-name-F* and the port number specified in *data-name-D* before sending messages.

If `Y` is specified for `DCCLTDELIVERYCHECK` of the client environment definition, the message delivery confirmation facility is used when a message is sent or received. In this case, the size of the message information sent before the message body is 11 bytes. After sending the 11-byte message information, TP1/Client returns control to the CUP.

Before executing `CBLDCCLS('ASMSEND ')`, make sure that you execute `CBLDCRPS('OPEN     ')` in which 4 or 16 is specified in *data-name-C*.

(3) Data area where the UAP sets values

- *data-name-A*

  Set `VALUE 'ASMSEND△'` as the request code for sending a message.

- *data-name-C*

  Specify whether to release the connection after the message is sent.

  `0`: After a message is sent, the connection is not released until `CBLDCRPS('CLOSE    ')` is executed (exception: an error occurs).

  `1`: After a message is sent, the connection is released. When the message delivery confirmation facility is being used, the connection is released after message information is received.

- *data-name-D*

  Specify the port number of the node to be connected when there is no connection and a connection must be established.

If `0` is specified, the statement accesses the contents of `DCSNDPORT` in the client environment definition acquired when `CBLDCRPS('OPEN     ')` was executed.

- *data-name-E*

  This argument takes effect when the message delivery confirmation facility is used. Specify the maximum time (in seconds) that the statement waits for response-only data to arrive. The value must be an integer from -1 to 65,535.

  If `-1` is specified:

  The statement waits indefinitely for response-only data.

  If `0` is specified:

  The statement does not wait for response-only data. If there is no message to be received, the statement returns a 02507 error.

  If any value from 1 to 65,535 is specified:

  The statement waits for a message, but returns a 02507 error if a message does not arrive within the specified number of seconds.

  If divided response-only data arrives, the statement repeats the receive processing until 11-byte response-only data arrives. The timeout specified by this argument is applied every time the statement attempts reception. If you want to use the value of this argument as the maximum response wait time for the client, specify the `00000002` option for `DCCLTOPTION` of the client environment definition.

- *data-name-F*

  Specify the host name of the node to be connected when no connection is established. You can specify a maximum of 63# characters for the host name. Make sure that the specified character string ends with a space character.

  If the specified character string begins with a space character, the statement references the contents of `DCSNDHOST` in the client environment definition acquired when `CBLDCRPS('OPEN     ')` was executed.

  You can also specify an IP address in decimal dot notation for the host name.

  #:

  If you specify `00000008` for `DCCLTOPTION` in the client environment definition, you can specify a maximum of 255 characters for the host name.

- *data-name-G*

  Set the length of the message to be sent.

- *data-name-H*

  Set the area that contains a message to be sent. The area must be larger than the length specified in *data-name-G*.

- *data-name-I*

  Set the client ID received by `CBLDCCLS('CLTIN    ')` or `CBLDCCLS('EXCLTIN ')`.

(4) Data area for which a value is returned

- *data-name-B*

  5-digit status code

(5) Status codes

Status code	Meaning
00000	Normal termination
02501	Invalid argument
02502	Possible causes are as follows: • `CBLDCRPS('OPEN     ')` has not been executed. • `CBLDCRPS('OPEN     ')` was executed, but neither 4 nor 16 was specified in *data-name-C*.
02504	Insufficient memory
02506	A network error occurred. The connection is released.
02507	A connection establishment request timed out. Alternatively, reception of response-only data timed out when the message delivery confirmation facility was being used. The connection is released.
02518	A system error occurred. If the error is a network error, the connection is released.
02538	Insufficient resource
02539	The host name is incorrect. Alternatively, a host name is not specified in either *data-name-F* or `DCSNDHOST`.
02541	A connection establishment request to the remote system was rejected.
02542	The connection was released by the remote system when the message delivery confirmation facility was being used.
02544	The client ID specified in *data-name-I* differs from the one received by `CBLDCCLS('CLTIN    ')` or `CBLDCCLS('EXCLTIN ')`.
02547	Port numbers that can be assigned automatically by the operating system are insufficient.
02548	An invalid message was received when the message delivery confirmation facility was being used. The connection is released.

Status code	Meaning
02584	Messages collided when the message delivery confirmation facility was being used. The connection is released.

(6) Notes

- If the remote system releases the connection when the statement sends a message to the remote system, depending on the length of the message, the statement might not be able to detect that the connection has been released. The following describes what occurs in this case according to the facility used.

  When the message assembly facility is used:

  If `CBLDCCLS('ASMSEND ')` fails to detect the release of a connection when it sends a message, a subsequent request statement might detect the release. This must be kept in mind when you create a CUP.

  When the message delivery confirmation facility is used:

  If `CBLDCCLS('ASMSEND ')` fails to detect the release of a connection when it sends a message, the statement detects the release when it receives response-only data.
- If the message assembly and message delivery confirmation facilities are used, short packets are used for sending and receiving. As a result, transmission processing might take more time. If more time might be required, specify `Y` for `DCCLTTCPNODELAY` of the client environment definition.

6.6.6 CBLDCCLS('ASMRECV ') - Receiving assembled messages

(1) Form

(a) In a multi-thread environment

■ **PROCEDURE DIVISION**

```
CALL 'CBLDCCLS' USING identifier-1 identifier-2 identifier-3
```

■ **DATA DIVISION**

```
01 identifier-1.
   02  data-name-A  PIC X(8) VALUE 'ASMRECV△'.
   02  data-name-B  PIC X(5).
   02  FILLER    PIC X(3).
   02  data-name-C  PIC S9(9) COMP VALUE ZERO.
   02  data-name-D  PIC S9(9) COMP.
01 identifier-2.
   02  data-name-E  PIC S9(9) COMP.
   02  data-name-F  PIC X(n).
01 identifier-3.
   02  data-name-G  PIC 9(9) COMP.
```

(b) In a single-thread environment

■ **PROCEDURE DIVISION**

```
CALL 'CBLDCCLT' USING identifier-1 identifier-2
```

■ **DATA DIVISION**

```
01 identifier-1.
   02  data-name-A  PIC X(8) VALUE 'ASMRECV△'.
   02  data-name-B  PIC X(5).
   02  FILLER    PIC X(3).
   02  data-name-C  PIC S9(9) COMP VALUE ZERO.
   02  data-name-D  PIC S9(9) COMP.
01 identifier-2.
   02  data-name-E  PIC S9(9) COMP.
   02  data-name-F  PIC X(n).
```

(2) Purpose

Uses the message assembly facility to receive messages. When this facility is used, the statement receives four-byte message information, and then receives data equivalent to the size set in the message information. The statement then stores the data in *data-name-F*. The four-byte message information is not stored in *data-name-F*. The length of the received message is stored in *data-name-E*. The message length stored in *data-name-E* does not include the message information length.

If `Y` is specified for `DCCLTDELIVERYCHECK` of the client environment definition, the statement also uses the message delivery confirmation facility. In this case, the statement receives 11-byte message information, and then receives data equivalent to the size set in the message information. The function then stores the data in *data-name-F*. The 11-byte message information is not stored in *data-name-F*. The length of the received message is stored in *data-name-E*. The message length stored in *data-name-E* does not include the message information length. If the received message information includes a response request, the function sends 11-byte message information, and then returns control to the CUP.

Before executing `CBLDCCLS('ASMRECV ')`, make sure that you execute `CBLDCRPS('OPEN    ')` in which 8 or 16 is specified in *data-name-C*.

(3) Data area where the UAP sets values

- *data-name-A*

  Set `VALUE 'ASMRECV∆'` as the request code for receiving a message.

- *data-name-C*

  Specify whether to release the connection after receiving a message.

  0: After a message is received, the connection is not released until `CBLDCRPS('CLOSE    ')` is executed (exception: an error occurs).

  2: After a message is received, the connection is released. If the message information received when the message delivery confirmation facility is being used includes a response request, the connection is released after message information has been sent.

- *data-name-D*

  Specify the maximum time (in seconds) that the statement waits for a message to arrive. The value must be an integer from -1 to 65,535.

  If `-1` is specified:

  The statement waits indefinitely for a message.

  If `0` is specified:

  The statement does not wait for a message. If there is no message to be received, the statement returns a 02507 error.

  If any value from 1 to 65,535 is specified:

  The statement waits for a message, but returns a 02507 error if no message arrives within the specified number of seconds.

If a divided message arrives, the statement repeats the receive processing until the entire message arrives. The timeout specified by this argument is applied every time the statement attempts a reception. If you want to use the value of this argument as the maximum response wait time for the client, specify the `00000002` option for `DCCLTOPTION` of the client environment definition.

- *data-name-E*

  Specify the length of the message to be received.

- *data-name-F*

  Specify the area where the received message will be stored. The area must be larger than the length specified in *data-name-E*.

- *data-name-G*

  Specify the client ID received by `CBLDCCLS('CLTIN    ')` or

`CBLDCCLS('EXCLTIN ').`

(4) Data area for which a value is returned

- *data-name-B*

  5-digit status code

- *data-name-E*

  The length of the received message is stored in this area. The stored length does not include the length of the message information.

  If a timeout occurs, the length of the data received before the timeout is stored.

- *data-name-F*

  The received message is stored in this area. The stored message does not include message information.

  If a timeout occurs, the data received before the timeout is stored.

(5) Status codes

Status code	Meaning
00000	Normal termination
02501	Invalid argument
02502	Possible causes are as follows: • `CBLDCRPS('OPEN    ')` has not been executed. • `CBLDCRPS('OPEN    ')` has been executed, but 8 or 16 is not specified for *data-name-C*.
02504	Insufficient memory
02506	A network error occurred. The connection is released.
02507	Message reception timed out. The connection is released.
02518	A system error occurred. If the error is a network error, the connection is released.
02538	Insufficient resource
02542	The remote system released the connection.
02544	The client ID specified for *data-name-G* differs from the one received by `CBLDCCLS('CLTIN   ')` or `CBLDCCLS('EXCLTIN ')`.
02546	The area prepared by the CUP was too small to receive the message from the remote system. The connection is released.
02548	An invalid message was received. The connection is released.

(6) Notes

- `CBLDCCLS('ASMRECV ')` returns control to the CUP in the following cases only:
    - When the statement has received message data equivalent to the length set in the message information
    - When a network error has occurred
    - When message reception has timed out
    - When the connection is released by the remote system
    - When the message storage area (specified by *data-name-F*) is too small to store the message sent by the remote system
    - When an invalid message has been received
- If the message assembly and message delivery confirmation facilities are used, short packets are used for sending and receiving. As a result, transmission processing might take time. If more time might be required, specify `Y` for `DCCLTTCPNODELAY` of the client environment definition.

6.7 Facility for receiving one-way messages from the server

6.7.1 CBLDCCLS('NOTIFY ') - Receiving one-way messages

(1) Form

(a) In a multi-thread environment

■ **PROCEDURE DIVISION**

```
CALL 'CBLDCCLS' USING identifier-1 identifier-2
```

■ **DATA DIVISION**

```
01 identifier-1.
   02 data-name-A PIC X(8)  VALUE 'NOTIFY '.
   02 data-name-B PIC X(5).
   02 FILLER     PIC X(3).
   02 data-name-C PIC S9(9) COMP VALUE ZERO.
   02 data-name-D PIC 9(4)  COMP.
   02 FILLER     PIC X(2).
   02 data-name-E PIC S9(9) COMP.
   02 data-name-F PIC X(64).
   02 data-name-G PIC X(8).
   02 FILLER     PIC 9(4)  COMP.
   02 FILLER     PIC X(2).
   02 data-name-I PIC X(256).
01 identifier-2.
   02 data-name-J PIC S9(9) COMP.
   02 data-name-K PIC X(n).
```

(b) In a single-thread environment

■ **PROCEDURE DIVISION**

```
CALL 'CBLDCCLT' USING identifier-1 identifier-2
```

■ **DATA DIVISION**

```
01 identifier-1.
   02 data-name-A PIC X(8)  VALUE 'NOTIFY '.
   02 data-name-B PIC X(5).
   02 FILLER     PIC X(3).
   02 data-name-C PIC S9(9) COMP VALUE ZERO.
   02 data-name-D PIC 9(4)  COMP.
   02 FILLER      PIC X(2).
   02 data-name-E PIC S9(9) COMP.
   02 data-name-F PIC X(64).
   02 data-name-G PIC X(8).
01 identifier-2.
   02 data-name-J PIC S9(9) COMP.
   02 data-name-K PIC X(n).
```

(2) Purpose

Waits for a message returned by the `CBLDCRPC('CLTSEND ')` request code issued on the server side. The program stops waiting for the message if a timeout occurs before receiving the message. The timeout is specified in *data-name-E*. On reception of the message, this program returns the status code, received message, host name of the message-originating server, and node identifier of the message-originating server, and control returns to the CUP. Before executing this program, you do not need to execute `CBLDCCLS('CLTIN   ')` and `CBLDCRPS('OPEN    ')`.

(3) Data area where the UAP sets values

- *data-name-A*

  Set `VALUE 'NOTIFY∆ ∆'` as a request code for receiving one-way notification messages.

- *data-name-C*

  Set 0.

- *data-name-D*

  Specify a client's port number between 5001 and 65535. Specify a unique port number for each process or thread when multiple processes or multiple threads are executed simultaneously on the same machine.

- *data-name-E*

  Specify a timeout value (seconds) between 0 and 65535. Value 0 causes an infinite wait.

- *data-name-I*

  Specify the path name of the client environment definition file. The path name must be specified with the full path or with a relative path from the current drive and the current directory. The following shows the order in which files are loaded when the path name is specified.

  - In TP1/Client/P

Client environment definition files are loaded in the following order:

1. The `BETRAN.INI` file in the Windows directory

2. The client environment definition file specified in *data-name-I*

The definitions in both the client environment definition file and the `BETRAN.INI` file take effect.

If the same definition is specified in each file with a different value, the value specified in the client environment definition file takes effect.

If neither the client environment definition file nor the `BETRAN.INI` file contains the necessary specification, TP1/Client/P uses the defaults.

- In TP1/Client/W

  All definitions specified in the environment variables will be invalid. TP1/Client/W uses the defaults for definitions that are not specified in the client environment definition file specified in *data-name-I*.

You can omit the path name by specifying a blank at the beginning of *data-name-I*. The following describes the operation when the path name is omitted.

- In TP1/Client/P

  TP1/Client/P uses the `BETRAN.INI` file in the Windows directory as the client environment definition file. If the `BETRAN.INI` file does not exist or if the contents of the definition file are invalid, TP1/Client/P uses the defaults.

- In TP1/Client/W

  TP1/Client/W uses the values specified in the environment variables. If an environment variable is not specified, TP1/Client/W uses the default.

The following describes operation when the client environment definition file specified in *data-name-I* does not exist or when the contents of the definition file are invalid.

- In TP1/Client/P

  TP1/Client/P uses the `BETRAN.INI` file in the Windows directory as the client environment definition file. If the `BETRAN.INI` file does not exist or if the contents of the definition file are invalid, TP1/Client/P uses the defaults.

- In TP1/Client/W

  TP1/Client/W uses the defaults. The values specified in the environment variables will be invalid.

- *data-name-J*

  Specify the length of the area (length of *data-name-K*) for storing a notification message from the server. Available values are in the range from 0 to `DCRPC_MAX_MESSAGE_SIZE`#.

  # If you specify 2 or a larger value for `DCCLTRPCMAXMSGSIZE` in the client environment definition, the value you specify is used rather than the value of `DCRPC_MAX_MESSAGE_SIZE` (1 megabyte).

- *data-name-K*

  Specify the area for storing a notification message from the server. This area must be larger than the length specified for *data-name-J*.

(4) Data area for which a value is returned

- *data-name-B*

  5-digit status code.

- *data-name-F*

  Host computer name for the server that notified the message. If resolution to a host name fails, the IP address is returned in the dotted decimal format.

- *data-name-G*

  Node identifier for the server that notified the message in the following format.

Node identifier (4 bytes)	Blank (4 bytes)

- *data-name-J*

  Notification message length from the server.

- *data-name-K*

  Notification message from the server.

(5) Status codes

Status code	Meaning
00000	Normal termination
02501	Invalid value for the data name. The request code (*data-name-A*) may be invalid.
02503	Unsuccessful initialization. Alternatively, the client environment definition is specified incorrectly.
02504	A necessary amount of buffer could not be allocated.
02506	Network error

Status code	Meaning
02507	Timeout occurred during reception of the message.
02518	System error
02535	Different versions
02546	The received message is too large for the CUP-provided area. The part that does not fit is truncated. Values have already been set for *data-name-F* and *data-name-G*.
02547	The specified port number is already used.
02548	Invalid message received
02549	The one-way message reception status was canceled by `CBLDCCLS('CANCEL  ')`. Values have already been set for *data-name-F*, *data-name-J*, and *data-name-K*.

(6) Note

Specify a unique port number in *data-name-D* for each process or thread when multiple processes or multiple threads are executed simultaneously on the same machine. Do not specify a port number for use by the operating system or other programs even if one can be specified in *data-name-D*. If you specify a port number in this case, response data might not be received correctly. The port numbers used by the operating system differ depending on the operating system. For details, see the documentation of your operating system.

6.7.2 CBLDCCLS('EXNACPT ') - Receiving one-way messages (for an extended host name)

(1) Form

(a) In a multi-thread environment

■ **PROCEDURE DIVISION**

```
CALL 'CBLDCCLS' USING identifier-1  identifier-2  identifier-3
```

■ **DATA DIVISION**

```
01 identifier-1.
   02 data-name-A PIC X(8)  VALUE 'EXNACPT '.
   02 data-name-B PIC X(5).
   02 FILLER    PIC X(3).
   02 data-name-C PIC S9(9) COMP VALUE ZERO.
   02 data-name-D PIC 9(4)  COMP.
   02 FILLER     PIC X(2).
   02 data-name-E PIC S9(9) COMP.
   02 data-name-F PIC X(8).
   02 data-name-G PIC X(n).
01 identifier-2.
   02 data-name-H PIC S9(9) COMP.
   02 data-name-I PIC X(n).
01 identifier-3.
   02 FILLER    PIC 9(9) COMP.
   02 FILLER    PIC 9(4) COMP.
   02 FILLER    PIC X(2).
   02 data-name-J PIC X(n).
```

(b) In a single-thread environment

■ **PROCEDURE DIVISION**

```
CALL 'CBLDCCLT' USING identifier-1 identifier-2
```

■ **DATA DIVISION**

```
01 identifier-1.
   02 data-name-A PIC X(8)  VALUE 'EXNACPT '.
   02 data-name-B PIC X(5).
   02 FILLER     PIC X(3).
   02 data-name-C PIC S9(9) COMP VALUE ZERO.
   02 data-name-D PIC 9(4)  COMP.
   02 FILLER     PIC X(2).
   02 data-name-E PIC S9(9) COMP.
   02 data-name-F PIC X(8).
   02 data-name-G PIC X(n).
01 identifier-2.
   02 data-name-H PIC S9(9) COMP.
   02 data-name-I PIC X(n).
```

(2) Purpose

Waits for a message returned by the `CBLDCRPC('CLTSEND ')` request code issued on the server side. The program stops waiting for the message if a timeout occurs before receiving the message. The timeout is specified in *data-name-E*. On reception of the message, this program returns the status code, received message, host name of the message-originating server, and node identifier of the message-originating server, and control returns to the CUP. Before executing this program, you do not need to execute `CBLDCCLS('CLTIN   ')` and `CBLDCRPS('OPEN    ')`.

Use this request statement when using the host name extension function.

(3) Data area where the UAP sets values

- *data-name-A*

  Set `VALUE 'EXNACPT△'` as a request code for receiving one-way notification messages.

- *data-name-C*

  Set `0`.

- *data-name-D*

  Specify a client's port number between 5001 and 65535. Specify a unique port number for each process or thread when multiple processes or multiple threads are executed simultaneously on the same machine.

- *data-name-E*

  Specify a timeout value (seconds) between 0 and 65535.

  Value 0 causes an infinite wait.

- *data-name-G*

  Specify an area of 64 bytes# or more for storing the host name of the server that sent a notification message.

  # This area must be larger than 255 bytes if you specify `00000008` for `DCCLTOPTION` in the client environment definition.

- *data-name-H*

  Specify the length of the area (length of *data-name-K*) for storing a notification message from the server. Available values are in the range from 0 to `DCRPC_MAX_MESSAGE_SIZE`#.

  # If you specify 2 or a larger value for `DCCLTRPCMAXMSGSIZE` in the client environment definition, the value you specify is used rather than the value of `DCRPC_MAX_MESSAGE_SIZE` (1 megabyte).

- *data-name-I*

  Specify the area for storing a notification message from the server. This area must be larger than the length specified for *data-name-H*.

- *data-name-J*

  Specify the path name of the client environment definition file. The path name must be specified with the full path or with a relative path from the current drive and the current directory. The following shows the order in which files are loaded when the path name is specified.

  - In TP1/Client/P

Client environment definition files are loaded in the following order:

1. The `BETRAN.INI` file in the Windows directory

2. The client environment definition file specified in *data-name-J*

The definitions in both the client environment definition file and the `BETRAN.INI` file take effect.

If the same definition is specified in each file with a different value, the value specified in the client environment definition file takes effect.

If neither the client environment definition file nor the `BETRAN.INI` file contains the necessary specification, TP1/Client/P uses the defaults.

- In TP1/Client/W

  All definitions specified in the environment variables will be invalid. TP1/Client/W uses the defaults for definitions that are not specified in the client environment definition file specified in *data-name-J*.

You can omit the path name by specifying a space character at the beginning of *data-name-J*. The following describes the operation when the path name is omitted.

- In TP1/Client/P

  TP1/Client/P uses the `BETRAN.INI` file in the Windows directory as the client environment definition file. If the `BETRAN.INI` file does not exist or if the contents of the definition file are invalid, TP1/Client/P uses the defaults.

- In TP1/Client/W

  TP1/Client/W uses the values specified in the environment variables. If an environment variable is not specified, TP1/Client/W uses the default.

The following describes operation when the client environment definition file specified in *data-name-J* does not exist or when the contents of the definition file are invalid.

- In TP1/Client/P

  TP1/Client/P uses the `BETRAN.INI` file in the Windows directory as the client environment definition file. If the `BETRAN.INI` file does not exist or if the contents of the definition file are invalid, TP1/Client/P uses the defaults.

- In TP1/Client/W

  TP1/Client/W uses the defaults. The values specified in the environment variables will be invalid.

(4) Data area for which a value is returned

- *data-name-B*

  5-digit status code.

- *data-name-F*

  Node identifier for the server that sent a notification message in the following format.

Node identifier (4 bytes)	Blank (4 bytes)

- *data-name-G*

  Host name of the server that sent a notification message. If resolution to a host name fails, the IP address is returned in the dotted decimal format.

- *data-name-H*

  Length of the notification message from the server.

- *data-name-I*

  Notification message from the server.

(5) Status codes

Status code	Meaning
00000	Normal termination
02501	Invalid value for the data name. The request code (*data-name-A*) may be invalid.
02503	Unsuccessful initialization. Alternatively, the client environment definition is specified incorrectly.
02504	A necessary amount of buffer could not be allocated.
02506	Network error
02507	A timeout occurred during message reception.
02518	System error
02535	Different versions
02546	The received message is too large for the CUP-provided area. The part that does not fit is truncated. Values have already been set for *data-name-F* and *data-name-G*.
02547	The specified port number is already used.
02548	Invalid message received

Status code	Meaning
02549	The one-way message reception status was canceled by `CBLDCCLS('CANCEL  ')` or `CBLDCCLS('EXNCANCL')`. Values have already been set for *data-name-G*, *data-name-H*, and *data-name-I*.

(6) Note

Specify a unique port number in *data-name-D* for each process or thread when multiple processes or multiple threads are executed simultaneously on the same machine. Do not specify a port number for use by the operating system or other programs even if one can be specified in *data-name-D*. If you specify a port number in this case, response data might not be received correctly. The port numbers used by the operating system differ depending on the operating system. For details, see the documentation of your operating system.

6.7.3 CBLDCCLS('CANCEL ') - Canceling one-way message wait state

(1) Form

(a) In a multi-thread environment

■ **PROCEDURE DIVISION**

```
CALL 'CBLDCCLS' USING identifier-1  identifier-2
```

■ **DATA DIVISION**

```
01 identifier-1.
   02 data-name-A PIC X(8)  VALUE 'CANCEL  '.
   02 data-name-B PIC X(5).
   02 FILLER     PIC X(3).
   02 data-name-C PIC S9(9) COMP VALUE ZERO.
   02 data-name-D PIC 9(4)  COMP.
   02 FILLER     PIC X(2).
   02 data-name-E PIC X(64).
   02 FILLER     PIC 9(4)  COMP.
   02 FILLER     PIC X(2).
   02 data-name-G PIC X(256).
01 identifier-2.
   02 data-name-H PIC S9(9) COMP.
   02 data-name-I PIC X(n).
```

(b) In a single-thread environment

■ **PROCEDURE DIVISION**

```
CALL 'CBLDCCLT' USING identifier-1 identifier-2
```

■ **DATA DIVISION**

```
01 identifier-1.
   02 data-name-A PIC X(8)  VALUE 'CANCEL  '.
   02 data-name-B PIC X(5).
   02 FILLER      PIC X(3).
   02 data-name-C PIC S9(9) COMP VALUE ZERO.
   02 data-name-D PIC 9(4)  COMP.
   02 FILLER      PIC X(2).
   02 data-name-E PIC X(64).
01 identifier-2.
   02 data-name-H PIC S9(9) COMP.
   02 data-name-I  PIC X(n).
```

(2) Purpose

Releases a wait state (enabled by `CBLDCCLS('NOTIFY  ')`) for receiving one-way messages.

When releasing the wait state, the program can issue a message specified for *data-name-I* to the CUP that awaits one-way messages.

(3) Data area where the UAP sets values

- *data-name-A*

  Set `VALUE 'CANCEL△△'` as the request code for indicating cancellation of a wait for a one-way message.

- *data-name-C*

  Set 0.

- *data-name-D*

  Specify the port number specified for a request to receive one-way messages between 5001 and 65535.

- *data-name-E*

  Specify the name of the host computer corresponding to the CUP that waits for a one-way message.

You can specify an IP address in decimal dot notation for the host name.

- *data-name-G*

  Specify the path name of the client environment definition file. The path name must be specified with the full path or with a relative path from the current drive and the current directory. The following shows the order in which files are loaded when the path name is specified.

  - In TP1/Client/P

    Client environment definition files are loaded in the following order:

    1. The `BETRAN.INI` file in the Windows directory

    2. The client environment definition file specified in *data-name-G*

    The definitions in both the client environment definition file and the `BETRAN.INI` file take effect.

    If the same definition is specified in each file with a different value, the value specified in the client environment definition file takes effect.

    If neither the client environment definition file nor the `BETRAN.INI` file contains the necessary specification, TP1/Client/P uses the defaults.

  - In TP1/Client/W

    All definitions specified in the environment variables will be invalid. TP1/Client/W uses the defaults for definitions that are not specified in the client environment definition file specified in *data-name-G*.

  You can omit the path name by specifying a space character at the beginning of *data-name-G*. The following describes the operation when the path name is omitted.

  - In TP1/Client/P

    TP1/Client/P uses the `BETRAN.INI` file in the Windows directory as the client environment definition file. If the `BETRAN.INI` file does not exist or if the contents of the definition file are invalid, TP1/Client/P uses the defaults.

  - In TP1/Client/W

    TP1/Client/W uses the values specified in the environment variables. If an environment variable is not specified, TP1/Client/W uses the default.

  The following describes operation when the client environment definition file specified in *data-name-G* does not exist or when the contents of the definition file are invalid.

  - In TP1/Client/P

TP1/Client/P uses the `BETRAN.INI` file in the Windows directory as the client environment definition file. If the `BETRAN.INI` file does not exist or if the contents of the definition file are invalid, TP1/Client/P uses the defaults.

- In TP1/Client/W

  TP1/Client/W uses the defaults. The values specified in the environment variables will be invalid.

■ *data-name-H*

Specify the message length (*data-name-I* length).

Available values range from 0 to `DCRPC_MAX_MESSAGE_SIZE`#. Specifying `0` notifies no messages to the CUP.

# If you specify 2 or a larger value for `DCCLTRPCMAXMSGSIZE` in the client environment definition, the value you specify is used rather than the value of `DCRPC_MAX_MESSAGE_SIZE` (1 megabyte).

■ *data-name-I*

Specify a message issued to the CUP.

(4) Data area for which a value is returned

■ *data-name-B*

5-digit status code.

(5) Status codes

Status code	Meaning
00000	Normal termination
02501	Invalid value for the data name. The request code (*data-name-A*) may be invalid.
02503	Initialization failed. Alternatively, the client environment definition is specified incorrectly.
02504	A necessary amount of buffer could not be allocated.
02506	Network error
02514	The CUP is not in the one-way message reception wait status.
02518	System error
02539	Invalid host computer name
02547	Port numbers that can be assigned automatically by the operating system are insufficient.
02554	No DLL name is defined in the client environment definition.

Status code	Meaning
02555	The specified DLL could not be loaded.
02556	An attempt was made to issue the request code not defined in the specified DLL.

6.7.4 CBLDCCLS('EXNCANCL') - Canceling one-way message wait state (for an extended host name)

(1) Form

(a) In a multi-thread environment

■ **PROCEDURE DIVISION**

```
CALL 'CBLDCCLS' USING identifier-1 identifier-2 identifier-3
```

■ **DATA DIVISION**

```
01 identifier-1.
   02 data-name-A  PIC X(8)  VALUE 'EXCANCEL'.
   02 data-name-B  PIC X(5).
   02 FILLER    PIC X(3).
   02 data-name-C  PIC S9(9) COMP VALUE ZERO.
   02 data-name-D  PIC 9(4)  COMP.
   02 FILLER    PIC X(2).
   02 data-name-E  PIC X(n).
01 identifier-2.
   02 data-name-F  PIC S9(9) COMP.
   02 data-name-G  PIC X(n).
01 identifier-3.
   02 FILLER    PIC 9(9) COMP.
   02 FILLER    PIC 9(4) COMP.
   02 FILLER    PIC X(2).
   02 data-name-H  PIC X(n).
```

(b) In a single-thread environment

■ **PROCEDURE DIVISION**

```
CALL 'CBLDCCLT' USING identifier-1 identifier-2
```

■ **DATA DIVISION**

```
01 identifier-1.
```

```
   02 data-name-A  PIC X(8)  VALUE 'EXCANCEL'.
   02 data-name-B  PIC X(5).
   02 FILLER    PIC X(3).
   02 data-name-C  PIC S9(9) COMP VALUE ZERO.
   02 data-name-D  PIC 9(4)  COMP.
   02 FILLER    PIC X(2).
   02 data-name-E  PIC X(n).
01 identifier-2.
   02 data-name-F  PIC S9(9) COMP.
   02 data-name-G  PIC X(n).
```

(2) Purpose

Releases a wait state (enabled by `CBLDCCLS('EXNACPT ')`) for receiving one-way messages.

When releasing the wait state, the program can issue a message specified for *data-name-I* to the CUP that awaits one-way messages.

Use this function when using the host name extension function.

(3) Data area where the UAP sets values

- *data-name-A*

  Set `VALUE 'EXCANCEL'` as the request code for indicating cancellation of a wait for a one-way message.

- *data-name-C*

  Set 0.

- *data-name-D*

  Specify the port number specified for a request to receive one-way messages between 5001 and 65535.

- *data-name-E*

  Specify the name of the host computer corresponding to the CUP that waits for a one-way message. You can specify a maximum of 63# characters for the host name. Terminate the character string with a blank.

  You can specify an IP address in decimal dot notation for the host name.

  # If you specify `00000008` for `DCCLTOPTION` in the client environment definition, you can specify a maximum of 255 characters for the host name.

- *data-name-F*

  Specify the message length. Available values are in the range from 0 to

`DCRPC_MAX_MESSAGE_SIZE`#.

If you specify `0`, no messages are sent to the CUP

# If you specify 2 or a larger value for `DCCLTRPCMAXMSGSIZE` in the client environment definition, the value you specify is used rather than the value of `DCRPC_MAX_MESSAGE_SIZE` (1 megabyte).

- *data-name-G*

  Specify the area for storing a notification message to be sent to the CUP. This area must be larger than the length specified for *data-name-F*.

- *data-name-H*

  Specify the path name of the client environment definition file. The path name must be specified with the full path or with a relative path from the current drive and the current directory. The following shows the order in which files are loaded when the path name is specified.

  - In TP1/Client/P

    Client environment definition files are loaded in the following order:

    1. The `BETRAN.INI` file in the Windows directory

    2. The client environment definition file specified in *data-name-H*

    The definitions in both the client environment definition file and the `BETRAN.INI` file take effect.

    If the same definition is specified in each file with a different value, the value specified in the client environment definition file takes effect.

    If neither the client environment definition file nor the `BETRAN.INI` file contains the necessary specification, TP1/Client/P uses the defaults.

  - In TP1/Client/W

    All definitions specified in the environment variables will be invalid. TP1/Client/W uses the defaults for definitions that are not specified in the client environment definition file specified in *data-name-H*.

  You can omit the path name by specifying a blank at the beginning of *data-name-H*. The following describes the operation when the path name is omitted.

  - In TP1/Client/P

    TP1/Client/P uses the `BETRAN.INI` file in the Windows directory as the client environment definition file. If the `BETRAN.INI` file does not exist or if the contents of the definition file are invalid, TP1/Client/P uses the defaults.

- In TP1/Client/W

  TP1/Client/W uses the values specified in the environment variables. If an environment variable is not specified, TP1/Client/W uses the default.

The following describes operation when the client environment definition file specified in *data-name-H* does not exist or when the contents of the definition file are invalid.

- In TP1/Client/P

  TP1/Client/P uses the `BETRAN.INI` file in the Windows directory as the client environment definition file. If the `BETRAN.INI` file does not exist or if the contents of the definition file are invalid, TP1/Client/P uses the defaults.

- In TP1/Client/W

  TP1/Client/W uses the defaults. The values specified in the environment variables will be invalid.

(4) Data area for which a value is returned

- *data-name-B*

  5-digit status code.

(5) Status codes

Status code	Meaning
00000	Normal termination
02501	Invalid value for the data name. The request code (*data-name-A*) may be invalid.
02503	Initialization failed. Alternatively, the client environment definition is specified incorrectly.
02504	A necessary amount of buffer could not be allocated.
02506	Network error
02514	The CUP is not in the one-way message reception wait status.
02518	System error
02539	Invalid host computer name
02547	Port numbers that can be assigned automatically by the operating system are insufficient.

6.7.5 CBLDCCLS('O-NOTIFY') - Start reception of one-way messages

(1) Form

(a) In a multi-thread environment

■ **PROCEDURE DIVISION**

```
CALL  'CBLDCCLS'  USING  identifier-1
```

■ **DATA DIVISION**

```
01  identifier-1.
  02  data-name-A  PIC  X(8)  VALUE 'O-NOTIFY'.
  02  data-name-B  PIC  X(5).
  02  FILLER  PIC  X(3).
  02  data-name-C  PIC  S9(9)  COMP VALUE ZERO.
  02  data-name-D  PIC  9(4)  COMP.
  02  FILLER  PIC  X(2).
  02  FILLER  PIC  9(4)  COMP.
  02  FILLER  PIC  X(2).
  02  data-name-F  PIC  9(9)  COMP.
  02  data-name-G  PIC  X(256).
```

(b) In a single-thread environment

■ **PROCEDURE DIVISION**

```
CALL  'CBLDCCLT'  USING  identifier-1
```

■ **DATA DIVISION**

```
01 identifier-1.
  02 data-name-A  PIC X(8) VALUE 'O-NOTIFY'.
  02 data-name-B  PIC X(5).
  02 FILLER PIC X(3).
  02 data-name-C PIC S9(9) COMP VALUE ZERO.
  02 data-name-D PIC 9(4) COMP.
  02 FILLER PIC X(2).
```

(2) Purpose

`CBLDCCLS('O-NOTIFY')` creates an environment for using the facility for receiving one-way messages from the server.

`CBLDCCLS('O-NOTIFY')` and `CBLDCCLS('C-NOTIFY')` are used in a pair.

(3) Data area where the UAP sets values

- *data-name-A*

  Set `VALUE 'O-NOTIFY'` as a request code for starting reception of one-way messages.

- *data-name-C*

  Set `0`.

- *data-name-D*

  Set the client's port number in the range from 5001 to 65535. Specify a unique port number for each process or thread when multiple processes or multiple threads are executed simultaneously on the same machine.

- *data-name-G*

  Specify the path name of the client environment definition file. The path name must be specified with the full path or with a relative path from the current drive and the current directory. The following shows the order in which files are loaded when the path name is specified.

  - In TP1/Client/P

    Client environment definition files are loaded in the following order:

    1. The `BETRAN.INI` file in the Windows directory

    2. The client environment definition file specified in *data-name-G*

    The definitions in both the client environment definition file and the `BETRAN.INI` file take effect.

    If the same definition is specified in each file with a different value, the value specified in the client environment definition file takes effect.

    If neither the client environment definition file nor the `BETRAN.INI` file contains the necessary specification, TP1/Client/P uses the defaults.

  - In TP1/Client/W

    All definitions specified in the environment variables will be invalid. TP1/Client/W uses the defaults for definitions that are not specified in the client environment definition file specified in *data-name-G*.

  You can omit the path name by specifying a blank at the beginning of *data-name-G*. The following describes the operation when the path name is omitted.

- In TP1/Client/P

  TP1/Client/P uses the `BETRAN.INI` file in the Windows directory as the client environment definition file. If the `BETRAN.INI` file does not exist or if the contents of the definition file are invalid, TP1/Client/P uses the defaults.

- In TP1/Client/W

  TP1/Client/W uses the values specified in the environment variables. If an environment variable is not specified, TP1/Client/W uses the default.

The following describes operation when the client environment definition file specified in *data-name-G* does not exist or when the contents of the definition file are invalid.

- In TP1/Client/P

  TP1/Client/P uses the `BETRAN.INI` file in the Windows directory as the client environment definition file. If the `BETRAN.INI` file does not exist or if the contents of the definition file are invalid, TP1/Client/P uses the defaults.

- In TP1/Client/W

  TP1/Client/W uses the defaults. The values specified in the environment variables will be invalid.

(4) Data area for which a value is returned

■ *data-name-B*

A five-digit status code is returned.

■ *data-name-F*

The one-way message reception ID is returned. Do not destroy the returned one-way message reception ID before `CBLDCCLS('C-NOTIFY')` is executed.

(5) Status codes

Status code	Meaning
00000	The program normally terminated.
02501	The value specified in the data area is incorrect.
02502	`CBLDCCLT('O-NOTIFY')` has already been executed. This status code is not returned if `CBLDCCLS('O-NOTIFY')` is executed.
02503	Initialization failed. Alternatively, the client environment definition is specified incorrectly.
02504	A necessary amount of buffer could not be allocated.

Status code	Meaning
02547	The specified port number has already been used.

(6) Notes

- After `CBLDCCLS('O-NOTIFY')` terminates normally, always execute `CBLDCCLS('C-NOTIFY')`. If `CBLDCCLS('C-NOTIFY')` is not executed, the resources used by `CBLDCCLS('O-NOTIFY')` may remain.
- Specify a unique port number in *data-name-D* for each process or thread when multiple processes or multiple threads are executed simultaneously on the same machine. Do not specify a port number for use by the operating system or other programs even if one can be specified in *data-name-D*. If you specify a port number in this case, response data might not be received correctly. The port numbers used by the operating system differ depending on the operating system. For details, see the documentation of your operating system.

6.7.6 CBLDCCLS('C-NOTIFY') - Terminate reception of one-way messages

(1) Form

(a) In a multi-thread environment

■ **PROCEDURE DIVISION**

```
CALL  'CBLDCCLS'  USING  identifier-1
```

■ **DATA DIVISION**

```
01  identifier-1.
  02  data-name-A  PIC  X(8)  VALUE 'C-NOTIFY'.
  02  data-name-B  PIC  X(5).
  02  FILLER  PIC  X(3).
  02  data-name-C  PIC  S9(9)  COMP VALUE ZERO.
  02  data-name-D  PIC  9(9)  COMP.
```

(b) In a single-thread environment

■ **PROCEDURE DIVISION**

```
CALL  'CBLDCCLT'  USING  identifier-1
```

■ **DATA DIVISION**

```
01  identifier-1.
  02  data-name-A  PIC  X(8)  VALUE 'C-NOTIFY'.
  02  data-name-B  PIC  X(5).
  02  FILLER  PIC  X(3).
  02  data-name-C  PIC  S9(9)  COMP VALUE ZERO.
```

(2) Purpose

`CBLDCCLS('C-NOTIFY')` deletes the environment for using the facility for receiving one-way messages from the server.

`CBLDCCLS('O-NOTIFY')` and `CBLDCCLS('C-NOTIFY')` are used in a pair.

(3) Data area where the UAP sets values

- *data-name-A*

  Set `VALUE 'C-NOTIFY'` as the request code for terminating reception of one-way messages.

- *data-name-C*

  Set `0`.

- *data-name-D*

  Specify the one-way message reception ID received by `CBLDCCLT('O-NOTIFY')`.

(4) Data area for which a value is returned

- *data-name-B*

  A five-digit status code is returned.

(5) Status codes

Status code	Meaning
00000	The program normally terminated.
02501	The value specified in the data area is incorrect.
02504	A necessary amount of buffer could not be allocated.
02544	The one-way message reception ID specified in *data-name-D* differs from that received by `CBLDCCLS('O-NOTIFY')`.

6.7.7 CBLDCCLS('A-NOTIFY') - Receive a one-way message

(1) Form

(a) In a multi-thread environment

■ **PROCEDURE DIVISION**

```
CALL  'CBLDCCLS'  USING  identifier-1 identifier-2
```

■ **DATA DIVISION**

```
01  identifier-1.
  02  data-name-A  PIC  X(8)  VALUE 'A-NOTIFY'.
  02  data-name-B  PIC  X(5).
  02  FILLER  PIC  X(3).
  02  data-name-C  PIC  S9(9)  COMP VALUE ZERO.
  02  data-name-D  PIC  S9(9)  COMP.
  02  data-name-E  PIC  X(64).
  02  data-name-F  PIC  X(8).
  02  data-name-G  PIC  9(9)  COMP.
01  identifier-2.
  02  data-name-H  PIC  S9(9)  COMP.
  02  data-name-I  PIC  X(n).
```

(b) In a single-thread environment

■ **PROCEDURE DIVISION**

```
CALL  'CBLDCCLT'  USING  identifier-1 identifier-2
```

■ **DATA DIVISION**

```
01  identifier-1.
  02  data-name-A  PIC  X(8)  VALUE 'A-NOTIFY'.
  02  data-name-B  PIC  X(5).
  02  FILLER  PIC  X(3).
  02  data-name-C  PIC  S9(9)  COMP VALUE ZERO.
  02  data-name-D  PIC  S9(9)  COMP.
  02  data-name-E  PIC  X(64).
  02  data-name-F  PIC  X(8).

01  identifier-2.
  02  data-name-H  PIC  S9(9)  COMP.
```

```
02  data-name-I  PIC  X(n).
```

(2) Purpose

Waits for a message returned by the `CBLDCRPC('CLTSEND ')` request code issued on the server side. The program stops waiting for the message if a timeout occurs before receiving the message. The timeout is specified in *data-name-D*. On reception of the message, this program returns the status code, received message, host name of the message-originating server, and node identifier of the message-originating server, and control returns to the CUP.

Before executing `CBLDCCLS('A-NOTIFY')`, always issue `CBLDCCLS('O-NOTIFY')`.

(3) Data area where the UAP sets values

- *data-name-A*

  Set `VALUE 'A-NOTIFY'` as the request code for receiving a one-way message.

- *data-name-C*

  Set `0`.

- *data-name-D*

  Set the timeout (in seconds) in the range from 0 to 65535. If 0 is set, a timeout does not occur.

- *data-name-G*

  Set the one-way message reception ID received by `CBLDCCLS('O-NOTIFY')`.

- *data-name-H*

  Specify the length of the area (length of *data-name-I*) for storing a notification message from the server. Available values are in the range from 0 to `DCRPC_MAX_MESSAGE_SIZE`#.

  # If you specify 2 or a larger value for `DCCLTRPCMAXMSGSIZE` in the client environment definition, the value you specify is used rather than the value of `DCRPC_MAX_MESSAGE_SIZE` (1 megabyte).

- *data-name-I*

  Specify the area for storing a notification message from the server. This area must be larger than the length specified for *data-name-H*.

(4) Data area for which a value is returned

- *data-name-B*

  A five-digit status code is returned.

- *data-name-E*

  The host name of the message-originating server is returned.

  If resolution to a host name fails, the IP address is returned in the dotted decimal format.

- *data-name-F*

  The node identifier of the message-originating server is returned. The node identifier is suffixed by a NULL character as shown below.

Node identifier (4 bytes)	Blank (4 bytes)

- *data-name-H*

  The notification message from the server is returned.

- *data-name-I*

  The length of the notification message from the server is returned.

(5) Status codes

Status code	Meaning
00000	The program normally terminated.
02501	The value specified in the data area is incorrect.
02502	`CBLDCCLS('O-NOTIFY')` has not been executed.
02504	A necessary amount of buffer could not be allocated.
02506	A network error occurred.
02507	A timeout occurred before a message arrived.
02518	A system error occurred.
02535	Versions do not match.
02544	The one-way message reception ID specified in *data-name-G* differs from that received by `CBLDCCLS('O-NOTIFY')`.
02546	The received message is too large for the CUP-provided area. The excess portion of the message is truncated. Values have already been set for *data-name-E* and *data-name-F*.
02548	An invalid message was received.
02549	The one-way message reception status was canceled by `CBLDCCLS('CANCEL  ')`. Values have already been set for *data-name-E*, *data-name-H*, and *data-name-I*.

6.7.8 CBLDCCLS('EXNCACPT') - Receive a one-way message (for an extended host name)

(1) Form

(a) In a multi-thread environment

■ **PROCEDURE DIVISION**

```
CALL 'CBLDCCLS' USING identifier-1 identifier-2 identifier-3
```

■ **DATA DIVISION**

```
01  identifier-1.
  02  data-name-A  PIC  X(8)  VALUE 'EXNCACPT'.
  02  data-name-B  PIC  X(5).
  02  FILLER  PIC  X(3).
  02  data-name-C  PIC  S9(9)  COMP VALUE ZERO.
  02  data-name-D  PIC  S9(9)  COMP.
  02  data-name-E  PIC  X(8).
  02  data-name-F  PIC  X(n).
01  identifier-2.
  02  data-name-G  PIC  S9(9)  COMP.
  02  data-name-H  PIC  X(n).
01  identifier-3.
  02  data-name-I  PIC  9(9)  COMP.
```

(b) In a single-thread environment

■ **PROCEDURE DIVISION**

```
CALL 'CBLDCCLT' USING identifier-1 identifier-2
```

■ **DATA DIVISION**

```
01  identifier-1.
  02  data-name-A  PIC  X(8)  VALUE 'EXNCACPT'.
  02  data-name-B  PIC  X(5).
  02  FILLER  PIC  X(3).
  02  data-name-C  PIC  S9(9)  COMP VALUE ZERO.
  02  data-name-D  PIC  S9(9)  COMP.
  02  data-name-E  PIC  X(8).
  02  data-name-F  PIC  X(n).
01  identifier-2.
```

```
02  data-name-G  PIC  S9(9)  COMP.
02  data-name-H  PIC  X(n).
```

(2) Purpose

Waits for a message returned by the `CBLDCRPC('CLTSEND ')` request code issued on the server side. The program stops waiting for the message if a timeout occurs before receiving the message. The timeout is specified in *data-name-D*. On reception of the message, this program returns the status code, received message, received message length, host name of the message-originating server, and node identifier of the message-originating server, and control returns to the CUP.

Before executing `CBLDCCLS('EXNCACPT')`, always issue `CBLDCCLS('O-NOTIFY')`.

Use this function when using the host name extension function.

(3) Data area where the UAP sets values

- *data-name-A*

  Set `VALUE 'EXNCACPT'` as the request code for receiving a one-way message.

- *data-name-C*

  Set 0.

- *data-name-D*

  Set the timeout (in seconds) in the range from 0 to 65535. If 0 is set, a timeout does not occur.

- *data-name-F*

  Specify an area of 64 bytes# or more for storing the host name of the server that sent a notification message.

  # This area must be larger than 255 bytes if you specify `00000008` for `DCCLTOPTION` in the client environment definition.

- *data-name-G*

  Specify the length of the area for storing a notification message from the server. Available values are in the range from 0 to `DCRPC_MAX_MESSAGE_SIZE`#.

  # If you specify 2 or a larger value for `DCCLTRPCMAXMSGSIZE` in the client environment definition, the value you specify is used rather than the value of `DCRPC_MAX_MESSAGE_SIZE` (1 megabyte).

- *data-name-H*

  Specify the area for storing a notification message from the server. This area must be larger than the length specified for *data-name-G*.

- *data-name-I*

  Specify the one-way message reception ID received by `CBLDCCLS('O-NOTIFY')`.

(4) Data area for which a value is returned

- *data-name-B*

  A five-digit status code is returned.

- *data-name-E*

  The node identifier of the message-originating server is returned in the following format:

Node identifier (4 bytes)	Blank (4 bytes)

- *data-name-F*

  The host name of the message-originating server is returned.

  If resolution to a host name fails, the IP address is returned in the dotted decimal format.

- *data-name-G*

  The length of the notification message from the server is returned.

- *data-name-H*

  The notification message from the server is returned.

(5) Status codes

Status code	Meaning
00000	Normal termination
02501	The value specified in the data area is incorrect.
02502	`CBLDCCLS('O-NOTIFY')` has not been executed.
02504	A necessary amount of buffer could not be allocated.
02506	Network error
02507	A timeout occurred during message reception.
02518	System error
02535	Versions do not match.

Status code	Meaning
02544	The one-way message reception ID specified in *data-name-I* differs from the one received by `CBLDCCLS('O-NOTIFY')`.
02546	The received message is too large for the CUP-provided area. The excess portion of the message is truncated. Values have already been set for *data-name-E* and *data-name-F*.
02548	An invalid message was received.
02549	The one-way message reception status was canceled by `CBLDCCLS('CANCEL   ')` or `CBLDCCLS('EXNCANCL')`. Values have already been set for *data-name-F*, *data-name-G*, and *data-name-H*.

6.8 Character code converter (When a code mapping table is not used)

The character code converter is only available for TP1/Client/P.

All request statements provided by the character code converter also operate correctly in a multi-thread environment.

6.8.1 CBLDCUTL ('CODECNV ') - Converting character codes

(1) Form

■ **PROCEDURE DIVISION**

```
CALL 'CBLDCUTL' USING identifier-1  identifier-2  identifier-3
```

■ **DATA DIVISION**

```
01 identifier-1.
   02 data-name-A PIC X(8)  VALUE 'CODECNV  '.
   02 data-name-B PIC X(5).
   02 FILLER     PIC X(3).
   02 data-name-C PIC S9(9) COMP VALUE ZERO.
   02 data-name-D PIC S9(9) COMP.
01 identifier-2.
   02 data-name-E PIC S9(9) COMP.
   02 data-name-F PIC X(n).
01 identifier-3.
   02 data-name-G PIC S9(9) COMP.
   02 data-name-H PIC X(n).
```

(2) Purpose

- Converts the character strings consisting of JIS code or Shift JIS code into character strings of EBCDIC/EBCDIK code or KEIS code.
- Converts the character strings consisting of EBCDIC/EBCDIK code or KEIS code into character strings of JIS code or Shift JIS code.

(3) Data area where the UAP sets values

■ *data-name-A*

Set `VALUE 'CODECNV△'` as a request code for character code conversion.

■ *data-name-C*

Specify the total number of options to be used as the conversion condition (conversion option).

`0`: Default (this condition is assumed if no option is specified).

- EBCDIK code is used.
- Two-byte spaces remain the same.
- The 1983 version of the KEIS code is used.
- An error occurs if an invalid code is found.
- A tab or control code is not identified to be single-byte. No shift code is available even for just the preceding or succeeding two-byte code if any.

`1`: EBCDIC code is used.

`2`: A two-byte space is converted to two spaces. This specification is valid only when the value of *data-name-D* is `1`.

`4`: The 1978 version of the KEIS code is used.

`8`: An invalid code is converted to a space.

`16`: A tab code is identified to be single-byte. A shift code is given to just the preceding or succeeding two-byte code if any.

`32`: A control code is identified to be single-byte. A shift code is given to just the preceding or succeeding two-byte code if any.

■ *data-name-D*

Specify the conversion method.

`1`: Converts the character strings consisting of JIS code or Shift JIS code into character strings of EBCDIC/EBCDIK code or KEIS code.

`2`: Converts the character strings consisting of EBCDIC/EBCDIK code or KEIS code into character strings of JIS code or Shift JIS code.

■ *data-name-E*

Specify the length of the character string to be converted. 1 to `DCRPC_MAX_MESSAGE_SIZE` can be specified.

■ *data-name-F*

Specify the character string to be converted.

■ *data-name-G*

Specify the size of the area that receives the converted character string.

■ *data-name-H*

Specify the area for storing the converted character string. This area must be larger than the length specified for *data-name-G*.

(4) Data area for which a value is returned

- *data-name-B*

  A status code is returned using a five-digit number.

- *data-name-G*

  The length of the converted character string is returned.

- *data-name-H*

  The converted character string is returned.

(5) Status codes

Status code	Meaning
00000	Normal termination
02501	Invalid value for the data name. The request code (*data-name-A*) may be invalid.
02504	Insufficient memory. The program returns this code also when the specified character length covers the first byte of a two-byte code that is contained in the character string to be converted.
02550	An invalid code is found in the character string.
02551	The length of the converted character string exceeds the area prepared by the CUP.

(6) Note

- When you specify `2` for *data-name-D* and `16` or `32` for *data-name-C*, you need to prepare data that contains single-byte tab or control codes.
- For details about code conversion specifications, see *A. Code Conversion Specifications*.

6.9 Character code converter (When a code mapping table is used)

The character code converter is only available for TP1/Client/P.

All request statements provided by the character code converter also operate correctly in a multi-thread environment.

6.9.1 CBLDCUTL('CNVOPN ') - Starting character code conversion

(1) Form

■ **PROCEDURE DIVISION**

```
CALL 'CBLDCUTL' USING unique-name-1  unique-name-1  unique-name-1
```

■ **DATA DIVISION**

```
01 unique-name-1.
   02 data-name-A PIC X(8)  VALUE 'CNVOPN  '.
   02 data-name-B PIC X(5).
   02 FILLER     PIC X(3).
   02 data-name-C PIC X(256)
   02 data-name-D PIC S9(9) COMP.
   02 data-name-E PIC 9(9) COMP.
```

(2) Purpose

Starts character code conversion to allocate a code mapping table to be used in the memory.

(3) Data area where the UAP sets values

- *data-name-A*

  Set `VALUE 'CNVOPN△△'` as the request code for indicating the start of character code conversion.

- *data-name-C*

  Set a blank.

- *data-name-D*

  Specify the conversion method.

  `1`: Links with CommuniNet for conversion.

  `0`: Performs conversion by operations without using a code mapping table.

(4) Data area for which a value is returned

- *data-name-B*

  A status code is returned using a five-digit number.

- *data-name-E*

  The handle of a character code conversion control table allocated on the memory is returned.

(5) Status codes

Status code	Meaning
00000	Normal termination
02501	Invalid value for the data name. The request code (*data-name-A*) may be invalid.
02504	Insufficient memory
02557	No code mapping table exists.
02558	This status does not support using the code mapping table. This status code is also returned if a code mapping table is not saved using the CommuniNet code mapping utility after the installation of CommuniNet.
02559	An I/O error occurred in the code mapping table.

(6) Notes

Specify three unique names 1 for the USING clause in the CALL statement.

- The use of this function requires a CommuniNet code mapping table. Before using this function, create a code mapping table using the CommuniNet code mapping utility.
- You cannot use a code mapping table unless you first save the table using the CommuniNet code mapping utility after the installation of CommuniNet. Before using this function, save a code mapping table using the CommuniNet code mapping utility.
- The filename of a CommuniNet code mapping table must be `CMAPEX.TBL`. Store the code mapping table under a Windows directory before using this function.
- The processing by the character code converter does not reflect the changes in the contents of a code mapping table changed by the CommuniNet code mapping utility during the use of this function.
- This function does not save error logs and UAP trace information.
- Issue the function for starting character code conversion (`CBLDCUTL('CNVOPN ')`) only once for code conversion (`CBLDCUTL('CNVEXEC  ')`). Do not issue

the function for starting character code conversion more than once to prevent memory shortage. If you issue two or more functions, issue one function for terminating character code conversion (`CBLDCUTL('CNVCLS  ')`) for each of the issued functions.

6.9.2 CBLDCUTL('CNVCLS ') - Terminating character code conversion

(1) Form

■ **PROCEDURE DIVISION**

```
CALL 'CBLDCUTL' USING unique-name-1  unique-name-1  unique-name-1
```

■ **DATA DIVISION**

```
01 unique-name-1.
   02 data-name-A PIC X(8)  VALUE 'CNVCLS  '.
   02 data-name-B PIC X(5).
   02 FILLER     PIC X(3).
   02 data-name-C PIC S9(9) COMP VALUE ZERO.
   02 data-name-D PIC S9(9) COMP.
```

(2) Purpose

Terminates character code conversion to free an area in the memory for allocating a code mapping table.

(3) Data area where the UAP sets values

■ *data-name-A*

Set `VALUE 'CNVCLSDD∆ ∆'` as the request code for indicating the termination of character code conversion.

■ *data-name-C*

Set `0`.

■ *data-name-D*

Specify the handle of the control table acquired by `CBLDCUTL ('CNVOPN  ')` for converting a character code.

(4) Data area for which a value is returned

■ *data-name-B*

A status code returned with only five-digit number.

(5) Status codes

Status code	Meaning
00000	Normal termination
02501	Invalid value for the data name. The request code (*data-name-A*) may be invalid.
02504	Insufficient memory

(6) Notes

Specify three unique names 1 for the USING clause in the CALL statement.

- The use of this function requires a CommuniNet code mapping table. Before using this function, create a code mapping table using the CommuniNet code mapping utility.
- You cannot use a code mapping table unless you save the table using the CommuniNet code mapping utility after the installation of CommuniNet. Before using this function, save a code mapping table using the CommuniNet code mapping utility.
- The filename of a CommuniNet code mapping table must be `CMAPEX.TBL`. Store the code mapping table under a Windows directory before using this function.
- The processing by the character code converter does not reflect changes made in a code mapping table by the CommuniNet code mapping utility during the use of this function.
- This function does not save error logs and UAP trace information.
- Issue the function for starting character code conversion (`CBLDCUTL('CNVOPN ')`) only once for code conversion (`CBLDCUTL('CNVEXEC  ')`). Do not issue the function for starting character code conversion more than once to prevent memory shortage. If you issue two or more functions, issue one function for terminating character code conversion (`CBLDCUTL('CNVCLS  ')`) for each of the issued functions.

6.9.3 CBLDCUTL('CNVEXEC') - Executing character code conversion

(1) Form

■ **PROCEDURE DIVISION**

CALL 'CBLDCUTL' USING *unique-name-1 unique-name-2 unique-name-3*

■ **DATA DIVISION**

```
01 unique-name-1.
   02 data-name-A PIC X(8)  VALUE 'CNVEXEC'.
   02 data-name-B PIC X(5).
   02 FILLER     PIC X(3).
   02 data-name-C PIC S9(9) COMP VALUE ZERO.
   02 data-name-D PIC S9(9) COMP.
   02 data-name-E PIC 9(9) COMP.
01 unique-name-2.
   02 data-name-F PIC S9(9) COMP.
   02 data-name-G PIC X(n).
01 unique-name-3.
   02 data-name-H PIC S9(9) COMP.
   02 data-name-I PIC X(n).
```

(2) Purpose

Executes the following character code conversion.

Converts character strings consisting of JIS or Shift JIS code into character strings of EBCDIC, EBCDIK or KEIS code. Converts character strings consisting of EBCDIC, EBCDIK or KEIS code into character strings of JIS or Shift JIS code.

(3) Data area where the UAP sets values

- *data-name-A*

  Set `VALUE 'CNVEXEC∆'` as the request code for indicating the execution of character code conversion.

- *data-name-C*

  Specify the conversion condition (conversion option) using the sum of the values of options to be used.

  `0`: Default (The default is applied without an option specified.)

  - The EBCDIK code set is used.
  - A two-byte space remains a two-byte space.
  - The 1983 version of the KEIS code is used.
  - If there is an invalid code, an error occurs.
  - A tab or control code is not recognized as a one-byte code. Even when there is a two-byte code immediately before or after the code, a shift code is not added.

  `1`: Uses the EBCDIC code.

  `2`: Converts a two-byte space into two one-byte spaces. This value is valid only when the value of *data-name-D* is 1.

`4`: Uses the 1978 version of the KEIS code.

`8`: Converts an invalid code into a space.

`16`: A tab code is identified to be single-byte. A shift code is given to just the preceding or succeeding two-byte code if any.

`32`: A control code is identified to be single-byte. A shift code is given to just the preceding or succeeding two-byte code if any.

- *data-name-D*

  Specify the conversion method.

  `1`: Converts a JIS or Shift-JIS character string into a EBCDIC, EBCDIK, or KEIS character string.

  `2`: Converts a EBCDIC, EBCDIK, or KEIS character string into a JIS or Shift-JIS character string.

- *data-name-E*

  Specify the handle of the control table acquired by `CBLDCUTL('CNVOPN  ')` for converting codes.

- *data-name-F*

  Specify the length of the character string to be converted. You can specify a value from 1 to `DCRPC_MAX_MESSAGE_SIZE`.

- *data-name-G*

  Specify the character string to be converted.

- *data-name-H*

  Specify the length of the area that receives the converted character string.

- *data-name-I*

  Specify the area for storing the converted character string. This area must be larger than the length specified for *data-name-H*.

(4) Data area for which a value is returned

- *data-name-B*

  A five-digit status code is returned.

- *data-name-H*

  The length of the converted character string is returned.

- *data-name-I*

The converted character string is returned.

(5) Status codes

Status code	Meaning
00000	Normal termination
02501	Invalid value for the data name. The request code (*data-name-A*) may be invalid.
02504	Insufficient memory. The program returns this code also when the control table contains an invalid handle value and when the specified character length covers the first byte of a two-byte code that is contained in the character string to be converted.
02550	A character string contains an invalid code.
02551	The length of the converted character string exceeds the length of the area prepared by the CUP.

(6) Notes

- The use of this function requires a CommuniNet code mapping table. Before using this function, create a code mapping table using the CommuniNet code mapping utility.
- You cannot use a code mapping table unless you first save the table using the CommuniNet code mapping utility after the installation of CommuniNet. Before using this function, save a code mapping table using the CommuniNet code mapping utility.
- The filename of a CommuniNet code mapping table must be `CMAPEX.TBL`. Store the code mapping table under a Windows directory before using this function.
- The processing by the character code converter does not reflect changes made in a code mapping table by the CommuniNet code mapping utility during the use of this function.
- This function does not save error logs and UAP trace information.
- Issue the function for starting character code conversion (`CBLDCUTL('CNVOPN ')`) only once for code conversion (`CBLDCUTL('CNVEXEC  ')`). Do not issue the function for starting character code conversion more than once to prevent memory shortage. If you issue two or more functions, issue a one function for terminating character code conversion (`CBLDCUTL('CNVCLS  ')`) for each of the issued functions.
- When you specify `2` for *data-name-D* and `16` or `32` for *data-name-C*, you need to prepare data that contains single-byte tab or control codes.

Chapter

7. Definition

This chapter describes the client environment definition.

In this chapter, C functions (`dc_`*xxx*`_`*xxx*`_s`) when calling the DLLs are used in explanations. If you use functions of the normal object library (`dc_`*xxx*`_`*xxx*) or COBOL, replace the C function names with the corresponding functions or COBOL request statements.

This chapter contains the following sections:

7.1 Overview

This section lists the client environment definition operands and describes definition conventions.

7.1.1 List of client environment definition operands

The following table lists the client environment definition operands.

Table 7-1: Client environment definition operands

No.	Operand	Description	Specifiable value
1	`DCNAMPORT`	Specifies the port number for the name server.	<unsigned integer> ((5001 to 65535)) <<10000>>
2	`DCHOST`	Specifies TP1/Server that operates as a gateway.	<character string>
3	`DCWATCHTIM`	Specifies the maximum time to wait for a response.	<unsigned integer> ((0 to 65535)) <<180>> (unit: seconds)
4	`DCCLTCONNECTTIMEOUT`	Specifies the maximum time to wait for a connection to be established.	<unsigned integer> ((0 to 65535)) <<0>> (unit: seconds)
5	`DCCLTTREXPTM`	Specifies the expiration time for a transaction branch.	<unsigned integer> ((0 to 65535)) (unit: seconds)
6	`DCCLTTREXPSP`	Specifies whether the monitoring time should include the time from when a transaction branch of the transactional RPC executing process uses the RPC facility to call another transaction branch until processing of the called branch terminates.	Y|N|F
7	`DCCLTTRWATTM`	Specifies the maximum time interval for a transaction inquiry response.	<unsigned integer> ((1 to 65535)) <<180>> (unit: seconds)
8	`DCCLTTRCPUTM`	Specifies the CPU monitoring time for a transaction branch.	<unsigned integer> ((0 to 65535)) (unit: seconds)
9	`DCCLTUTTRCMT`	When the online tester functionality is used to start a transaction from the CUP, this operand specifies whether to commit or roll back the transaction at a synchronous point.	Y|<<N>>

No.	Operand	Description	Specifiable value
10	`DCRCVPORT`	Specifies the receive port number for the CUP.	<unsigned integer> ((1 to 65535)) <<11000>>
11	`DCSNDHOST`	Specifies the name of the node to be connected.	<character string>
12	`DCSNDPORT`	Specifies the port number of the node to be connected.	<unsigned integer> ((1 to 65535)) <<12000>>
13	`DCSOCKOPENATRCV`	When the TCP/IP communication facility is used and a single connection is used for both sending and receiving, this operand specifies when to open the receive socket (the time at which to start waiting for a connection from the send destination).	Y|<<N>>
14	`DCCLTDELIVERYCHECK`	Specifies whether to use the message delivery confirmation facility.	Y|<<N>>
15	`DCUTOKEY`	Specifies the test user ID.	<1 to 4 alphanumeric characters>
16	`DCCACHE`	Specifies the number of areas for temporarily storing service information.	<unsigned integer> ((2 to 10240)) <<8>>
17	`DCCLTCACHETIM`	Specifies the effective period for the temporarily stored service information.	<unsigned integer> ((0 to 65535)) <<30>> (unit: seconds)
18	`DCCLTLOADBALANCE`	When a multi-node server is used, this operand specifies whether to use the inter-node load-balancing facility. This facility evaluates the load status of each node in TP1/Client when an RPC is performed and distributes processing to the server with the least load.	Y|<<N>>
19	`DCCLTSERVICEGROUPLIST`	Specifies the name of the file that defines the correspondence between service groups and RPC entry points.	<character string>
20	`DCCLTCONNECTRETRY`	Specifies the maximum number of attempts for establishing a connection.	<unsigned integer> ((0 to 255)) <<0>>
21	`DCSCDDIRECT`	Specifies whether to use the functionality that directly sends inquires about service information to the schedule service without sending inquiries about the information to the TP1/Server name service (RPCs that do not use the name service).	Y|<<N>>

No.	Operand	Description	Specifiable value
22	`DCSCDPORT`	Specifies the port number of the schedule service.	<unsigned integer> ((5001 to 65535))
23	`DCCLTDATACOMP`	Specifies whether to use the data compression facility.	Y|<<N>>
24	`DCEXTENDFUNCTION`	Specifies the level to which the RPC service functionality is extended.	<unsigned hexadecimal number> ((00000000 to 00000001)) <<00000000>>
25	`DCCLTINQUIRETIME`	Specifies the maximum time interval in permanent connection.	<unsigned integer> ((0 to 1048575)) (unit: seconds)
26	`DCCLTPORT`	Specifies the port number of the client extended service.	<unsigned integer> ((5001 to 65535))
27	`DCCLTDCCMHOST`	When a DCCM3 logical terminal is asked to establish a permanent connection, this operand specifies the host name of the logical terminal.	Host name of a DCCM3 logical terminal
28	`DCCLTDCCMPORT`	Specifies the port number of a DCCM3 logical terminal.	<unsigned integer> ((1 to 65535)) <<30000>>
29	`DCCLTXATMI`	Specifies whether to use the XATMI interface for communication.	Y|<<N>>
30	`DCWATCHTIMINHERIT`	When transaction control and connection control are performed, this operand specifies whether the client extended service inherits the maximum CUP response wait time.	Y|<<N>>
31	`DCCLTDELAY`	Specifies the maximum communication delay time.	<unsigned integer> ((0 to 65535)) <<0>> (unit: seconds)
32	`DCCLTCUPSNDHOST`	Specifies the CUP send host.	<character string>
33	`DCCLTCUPRCVPORT`	Specifies the receive port number for the CUP.	<unsigned integer> ((5001 to 65535))
34	`DCCLTRAPHOST`	Specifies the host name and port number of the TP1/Server's RAP-processing listener or of the DCCM3 logical terminal.	RAP-processing listener or DCCM3 logical terminal

No.	Operand	Description	Specifiable value
35	`DCCLTRAPAUTOCONNECT`	Specifies whether to enable automatic establishment of a connection between the CUP and RAP-processing server or DCCM3 logical terminal.	Y|<<N>>
36	`DCCLTTRSTATISITEM`	Specifies the items for which transaction branch statistics are to be acquired.	*statistics-item*`[,`*statistics-item*`]`...
37	`DCCLTTROPTIITEM`	Specifies optimization items for enhancing the performance of a global transaction consisting of two or more user servers.	*transaction-optimization-item*`[,`*transaction-optimization-item*`]`...
38	`DCCLTTRWATCHTIME`	Specifies the maximum time to wait for communication during transaction synchronous point processing.	<unsigned integer> ((1 to 65535)) (unit: seconds)
39	`DCCLTTRRBINFO`	If a transaction branch is rolled back, this operand specifies whether to log the information about why the transaction branch was rolled back.	no|self|remote|all
40	`DCCLTTRLIMITTIME`	Specifies the maximum time for executing a transaction branch.	<unsigned integer> ((0 to 65535)) (unit: seconds)
41	`DCCLTTRRBRCV`	Specifies whether to receive notice of the completion of a rollback after sending rollback directions to a destination transaction branch of an RPC.	Y|N
42	`DCCLTTRRECOVERYTYPE`	Specifies a method for processing a transaction synchronous point when a UAP error occurs.	type1|type2|type3
43	`DCWATCHTIMRPCINHERIT`	Specifies whether the server should inherit the maximum wait time for a CUP response.	Y|<<N>>
44	`DCSYSWATCHTIM`	Specifies the maximum time that OpenTP1 control waits for a response.	<unsigned integer> ((0 to 65535)) <<maximum response wait time>> (unit: seconds)
45	`DCCLTAUTHENT`	Specifies whether to enable user authentication when the `dc_clt_cltin_s` function is issued.	<<Y>>|N
46	`DCCLTCONNECTINF`	Specifies terminal identification information.	Terminal identification information

No.	Operand	Description	Specifiable value
47	`DCSCDMULTI`	Specifies whether to use the multi-scheduler facility.	Y|<<N>>
48	`DCSCDMULTICOUNT`	Specifies the number of multi-scheduler daemon processes.	<unsigned integer> ((1 to 4096)) <<1>>
49	`DCHOSTSELECT`	Specifies whether to select the gateway TP1/Server at random.	Y|<<N>>
50	`DCSCDLOADPRIORITY`	Specifies whether to prioritize distribution of the load on the gateway TP1/Server that receives service requests.	Y|<<N>>
51	`DCCLTONLYTHISNODE`	Specifies whether to execute services on the specified node when the `dc_rpc_call_to_s` function is issued.	Y|<<N>>
52	`DCCLTNOSERVER`	Specifies whether the environment being used is an environment in which TP1/Client never communicates with TP1/Server.	Y|<<N>>
53	`DCHOSTCHANGE`	Specifies whether to switch the gateway TP1/Server to another TP1/Server if TP1/Client receives an error reply from the gateway TP1/Server in one of the following cases: (1) when a service request is issued (when the `dc_rpc_call_s` function is executed), (2) when the schedule service is being started, or (3) when the schedule service is being terminated.	<<Y>>|N
54	`DCCLTOPTION`	Specifies the option for extending the functionality of the client.	<unsigned hexadecimal number> ((00000000 to 00000008)) <<00000000>>
55	`DCCLTNAMEXTEND`	Increases the number of service information items acquired and managed by clients when multi-node servers are used.	<<0>>|1
56	`DCTRCPATH`	Specifies the path to the directory where trace files are created.	<character string>
57	`DCTRCERR`	Specifies the size of error log files.	<unsigned integer> ((0 to 1073741824)) <<4096>> (unit: bytes)

No.	Operand	Description	Specifiable value
58	`DCTRCUAP`	Specifies the size of UAP trace files.	<unsigned integer> ((4096 to 1073741824)) <<trace information is not output>> (unit: bytes)
59	`DCTRCSOC`	Specifies the size of socket trace files.	<unsigned integer> ((4096 to 1073741824)) <<trace information is not output>> (unit: bytes)
60	`DCTRCSOCSIZE`	Specifies the data size of a socket trace to be output.	<unsigned integer> ((64 to 4096)) <<256>> (unit: bytes)
61	`DCTRCMDL`	Specifies the size of module trace files.	<unsigned integer> ((4096 to 1073741824)) <<trace information is not output>> (unit: bytes)
62	`DCCLTPRFINFOSEND`	Specifies whether to send identification information for the performance verification trace to TP1/Server.	Y|<<N>>
63	`DCCLTRPCMAXMSGSIZE`	Specifies the maximum length of a message that can be sent or received when an RPC is used.	<unsigned integer> ((1 to 8)) <<1>> (unit: MB)
64	`DCCLTRECVBUFSIZE`	Specifies the TCP/IP receive buffer size.	<unsigned integer> ((8192 to 1048576)) (unit: bytes)
65	`DCCLTSENDBUFSIZE`	Specifies the TCP/IP send buffer size.	<unsigned integer> ((8192 to 1048576)) (unit: bytes)
66	`DCCLTTCPNODELAY`	Specifies whether to disable the Nagle algorithm.	Y|<<N>>
67	`DCCLTBACKLOGCOUNT`#1	Specifies the number of queues for storing connection establishment requests.	<unsigned integer> ((0 to 4096)) <<0>>
68	`dcselint`#2	Specifies the interval for checking whether a response has been received.	<unsigned integer> ((0 to 65535)) <<100>> (unit: milliseconds)

#1:

This operand can be used in TP1/Client/W only.

#2:

This operand can be used in TP1/Client/P only.

7.1.2 Definition conventions

This subsection describes the symbols that are used for describing definitions.

The syntax symbols, attribute symbols, and syntax element symbols are not included in the actual definitions.

(1) Syntax symbols

The following table lists the symbols that describe the syntax.

Syntax symbol	Meaning
[ ]	The item between brackets can be omitted. Example: `CLTRESDN [/U]` Specify `CLTRESDN` or `CLTRESDN /U`.
|	One of the items separated by vertical bars can be selected. Example: `DCCLTTREXPSP=Y|N` Specify `DCCLTTREXPSP=Y` or `DCCLTTREXPSP=N`.
...	This symbol indicates a description is omitted. The item immediately before this symbol can be specified more than once consecutively. Example: *host-name* [:*port-number*][,*host-name* [:*port-number*],...] "*host-name*[:*port-name*]" can be specified more than once consecutively.

(2) Attribute symbols

The following table lists the symbols that describe the range of user-specified values.

Attribute symbol	Meaning
~	Attributes of the user-specified value are inserted after this symbol.
<< >>	Default for the user-specified value
< >	Syntax element symbol for the user-specified value
(( ))	Specification range of the user-specified value

(3) Syntax element symbols

The following table lists the symbols that describe the contents of user-specified values.

Syntax element symbol	Meaning
<alphabetic character>	Alphabetic character (A to Z, a to z) and underscore (_)

Syntax element symbol	Meaning
<alphanumeric>	Alphabetic characters and numbers (0 to 9)
<alphabetic character and symbol>	Alphabetic characters (A to Z, a to z), #, @, and \
<unsigned integer>	Numbers 0 to 9
<unsigned hexadecimal number>	Numbers 0 to 9, A to F, a to f
<symbolic name>	Alphabetic character or symbol and a number (the beginning character must be an alphabetic character or symbol)
<character string>	Any character(s)
<path name>	Symbolic name, /, and period (.) (The path name depends on the operating system used.)

7.2 Definition details

The environment variables shown below set client environment definition. The method used to set the environment variables varies depending on the operating system of the client machine. In TP1/Client/W, the method also depends on the shell.

7.2.1 TP1/Client/W format

(1) sh (Bourne shell)

```
$ DCNAMPORT=name-service-port-number
$ DCHOST=TP1/Server-as-a-gateway
$ DCWATCHTIM=maximum-response-wait-time
$ DCCLTCONNECTTIMEOUT=timeout-for-establishing-connection
$ DCCLTTREXPTM=expiry-time-in-transaction-branch
$ DCCLTTREXPSP=Y|N|F
$ DCCLTTRWATTM=maximum-time-interval-in-transaction-inquiry-response
$ DCCLTTRCPUTM=CPU-monitoring-time-in-transaction-branch
$ DCCLTUTTRCMT=Y|N
$ DCRCVPORT=CUP-receive-port-number
$ DCSNDHOST=connected-node-name
$ DCSNDPORT=connected-port-number
$ DCSOCKOPENATRCV=Y|N
$ DCCLTDELIVERYCHECK=Y|N
$ DCUTOKEY=test-user-ID
$ DCCACHE=number-of-areas-for-temporarily-storing-service-information
$ DCCLTCACHETIM=expiration-of-temporarily-stored-service-information
$ DCCLTLOADBALANCE=Y|N
$ 
DCCLTSERVICEGROUPLIST=file-defining-correspondence-between-service-groups-
and-RPCs
$ DCCLTCONNECTRETRY=retry-count-for-connection-establishment
$ DCSCDDIRECT=Y|N
$ DCSCDPORT=schedule-service-port-number
$ DCCLTDATACOMP=Y|N
$ DCEXTENDFUNCTION=facility-extension-level-of-RPC-service
$ DCCLTINQUIRETIME=maximum-time-interval-in-permanent-connection
$ DCCLTPORT=client-extended-service-port-number
$ DCCLTDCCMHOST=DCCM3-logical-terminal-host-name
$ DCCLTDCCMPORT=DCCM3-logical-terminal-port-number
$ DCCLTXATMI=Y|N
$ DCWATCHTIMINHERIT=Y|N
$ DCCLTDELAY=maximum-communication-delay-time
$ DCCLTCUPSNDHOST=CUP-send-host
$ DCCLTCUPRCVPORT=port-number-used-for-CUP-reception
$ DCCLTRAPHOST=RAP-processing-listener or DCCM3-logical-terminal
$ DCCLTRAPAUTOCONNECT=Y|N
$ DCCLTTRSTATISITEM=statistics-item
$ DCCLTTROPTIITEM=transaction-optimization-item
$
DCCLTTRWATCHTIME=maximum-communication-wait-time-during-synchronous-tran
saction-point-processing
$ DCCLTTRRBINFO=no|self|remote|all
$ DCCLTTRLIMITTIME=maximum-transaction-branch-execution-time
$ DCCLTTRRBRCV=Y|N
$ DCCLTTRRECOVERYTYPE=type1|type2|type3
$ DCWATCHTIMRPCINHERIT=Y|N
$ DCSYSWATCHTIM=maximum-response-wait-time-for-OpenTP1-control
$ DCCLTAUTHENT=Y|N
$ DCCLTCONNECTINF=terminal-identification-information
$ DCSCDMULTI=Y|N
$ DCSCDMULTICOUNT=number-of-multi-scheduler-daemon-processes
$ DCHOSTSELECT=Y|N
$ DCSCDLOADPRIORITY=Y|N
$ DCCLTONLYTHISNODE=Y|N
$ DCCLTNOSERVER=Y|N
$ DCHOSTCHANGE=Y|N
$ DCCLTOPTION=client's-extension-option
$ DCCLTNAMEXTEND=0|1
$
DCCLTBACKLOGCOUNT=number-of-queues-for-storing-connection-establishment-req
uests
$ DCTRCPATH=trace-output-directory
$ DCTRCERR=error-log-size
$ DCTRCUAP=UAP-trace-file-size
$ DCTRCSOC=socket-trace-file-size
$ DCTRCSOCSIZE=socket-trace-record-size
$ DCTRCMDL=module-trace-file-size
$ DCCLTPRFINFOSEND=Y|N
$ DCCLTRPCMAXMSGSIZE=maximum-RPC-message-length
$ DCCLTRECVBUFSIZE=TCP/IP-receive-buffer-size
$ DCCLTSENDBUFSIZE=TCP/IP-send-buffer-size
$ DCCLTTCPNODELAY=Y|N
$ export DCNAMPORT DCHOST DCWATCHTIM DCCLTCONNECTTIMEOUT
         DCCLTTREXPTM DCCLTTREXPSP DCCLTTRWATTM DCCLTTRCPUTM
         DCCLTUTTRCMT DCRCVPORT DCSNDHOST DCSNDPORT DCUTOKEY
         DCCACHE DCCLTCACHETIM DCCLTLOADBALANCE
         DCCLTSERVICEGROUPLIST DCCLTCONNECTRETRY DCSCDDIRECT
         DCSCDPORT DCCLTDATACOMP DCEXTENDFUNCTION
         DCCLTINQUIRETIME DCCLTPORT DCCLTDCCMHOST
```

```
                  DCCLTDCCMPORT DCCLTXATMI DCWATCHTIMINHERIT
                  DCCLTDELAY DCCLTCUPRCVPORT DCCLTRAPHOST
                  DCCLTRAPAUTOCONNECT
                  DCCLTTRSTATISITEM DCCLTTROPTIITEM DCCLTTRWATCHTIME
                  DCCLTTRRBINFO DCCLTTRLIMITTIME DCCLTTRRBRCV
                  DCCLTTRRECOVERYTYPE DCWATCHTIMRPCINHERIT DCSYSWATCHTIM
                 DCSOCKOPENATRCV DCCLTAUTHENT DCCLTCONNECTINF DCSCDMULTI
                  DCSCDMULTICOUNT DCHOSTSELECT DCSCDLOADPRIORITY
                DCCLTONLYTHISNODE DCCLTNOSERVER DCHOSTCHANGE DCCLTOPTION
                  DCCLTNAMEXTEND DCCLTBACKLOGCOUNT DCTRCPATH DCTRCERR
         DCTRCUAP
                  DCTRCSOC DCTRCSOCSIZE DCTRCMDL DCCLTRPCMAXMSGSIZE
                  DCCLTRECVBUFSIZE DCCLTSENDBUFSIZE DCCLTTCPNODELAY
```

(2) csh (C shell)

`% setenv DCNAMPORT` *name-service-port-number*
`% setenv DCHOST` *TP1/Server-as-a-gateway*
`% setenv DCWATCHTIM` *maximum-response-wait-time*
`% setenv DCCLTCONNECTTIMEOUT` *timeout-for-establishing-connection*
`% setenv DCCLTTREXPTM` *expiry-time-in-transaction-branch*
`% setenv DCCLTTREXPSP Y|N|F`
`% setenv DCCLTTRWATTM` *maximum-time-interval-in-transaction-inquiry-response*
`% setenv DCCLTTRCPUTM` *CPU-monitoring-time-in-transaction-branch*
`% setenv DCCLTUTTRCMT Y|N`
`% setenv DCRCVPORT` *CUP-receive-port-number*
`% setenv DCSNDHOST` *connected-node-name*
`% setenv DCSNDPORT` *connected-port-number*
`% setenv DCSOCKOPENATRCV Y|N`
`% setenv DCCLTDELIVERYCHECK Y|N`
`% setenv DCUTOKEY` *test-user-ID*
`% setenv DCCACHE` *number-of-areas-for-temporarily-storing-service-information*
`% setenv DCCLTCACHETIM` *expiration-of-temporarily-stored-service-information*
`% setenv DCCLTLOADBALANCE Y|N`
`% setenv DCCLTSERVICEGROUPLIST`
file-defining-correspondence-between-service-groups-and-RPCs
`% setenv DCCLTCONNECTRETRY` *entry-count-for-connection-establishment*
`% setenv DCSCDDIRECT Y|N`
`% setenv DCSCDPORT` *schedule-service-port-number*
`% setenv DCCLTDATACOMP Y|N`
`% setenv DCEXTENDFUNCTION` *facility-extension-level-of-RPC-service*
`% setenv DCCLTINQUIRETIME` *maximum-time-interval-in-permanent-connection*
`% setenv DCCLTPORT` *client-extended-service-port-number*
`% setenv DCCLTDCCMHOST` *DCCM3-logical-terminal-host-name*
`% setenv DCCLTDCCMPORT` *DCCM3-logical-terminal-port-number*
`% setenv DCCLTXATMI Y|N`

```
% setenv DCWATCHTIMINHERIT Y|N
% setenv DCCLTDELAY maximum-communication-delay-time
% setenv DCCLTCUPSNDHOST CUP-send-host
% setenv DCCLTCUPRCVPORT port-number-used-for-CUP-reception
% setenv DCCLTRAPHOST RAP-processing-listener or DCCM3-logical-terminal
% setenv DCCLTRAPAUTOCONNECT Y|N
% setenv DCCLTTRSTATISITEM statistics-item
% setenv DCCLTTROPTIITEM transaction-optimization-item
% setenv DCCLTTRWATCHTIME
maximum-communication-wait-time-during-synchronous-transaction-point-processi
ng
% setenv DCCLTTRRBINFO no|self|remote|all
% setenv DCCLTTRLIMITTIME maximum-transaction-branch-execution-time
% setenv DCCLTTRRBRCV Y|N
% setenv DCCLTTRRECOVERYTYPE type1|type2|type3
% setenv DCWATCHTIMRPCINHERIT Y|N
% setenv DCSYSWATCHTIM maximum-response-wait-time-for-OpenTP1-control
% setenv DCCLTAUTHENT Y|N
% setenv DCCLTCONNECTINF terminal-identification-information
% setenv DCSCDMULTI Y|N
% setenv DCSCDMULTICOUNT number-of-multi-scheduler-daemon-processes
% setenv DCHOSTSELECT Y|N
% setenv DCSCDLOADPRIORITY Y|N
% setenv DCCLTONLYTHISNODE Y|N
% setenv DCCLTNOSERVER Y|N
% setenv DCHOSTCHANGE Y|N
% setenv DCCLTOPTION client's-extension-option
% setenv DCCLTNAMEXTEND 0|1
% setenv DCCLTBACKLOGCOUNT
number-of-queues-for-storing-connection-establishment-requests
% setenv DCTRCPATH trace-output-directory
% setenv DCTRCERR error-log-size
% setenv DCTRCUAP UAP-trace-file-size
% setenv DCTRCSOC socket-trace-file-size
% setenv DCTRCSOCSIZE socket-trace-record-size
% setenv DCTRCMDL module-trace-file-size
% setenv DCCLTPRFINFOSEND Y|N
% setenv DCCLTRPCMAXMSGSIZE maximum-RPC-message-length
```

7.2.2 TP1/Client/P format

```
[betran]#
dcnamport=name-service-port-number
dchost=TP1/Server-as-a-gateway
dcwatchtim=maximum-wait-time
dccltconnecttimeout=maximum-connection-establishment-monitoring-time
```

```
dcclttrexptm=expiry-time-in-transaction-branch
dcclttrexpsp=y|n|f
dcclttrwattm=maximum-time-interval-in-transaction-inquiry-response
dcclttrcputm=CPU-monitoring-time-in-transaction-branch
dccltuttrcmt=y|n
dcrcvport=CUP-receive-port-number
dcsndhost=connected-node-name
dcsndport=connected-port-number
dcsockopenatrcv=y|n
dccltdeliverycheck=y|n
dcutokey=test-user-ID
dccache=number-of-areas-for-temporarily-storing-service-information
dccltcachetim=expiration-of-temporarily-stored-service-information
dccltloadbalance=y|n
dccltservicegrouplist=file-defining-correspondence-between-service-groups-
and-RPCs
dccltconnectretry=retry-count-for-connection-establishment
dcscddirect=y|n
dcscdport=schedule-service-port-number
dccltdatacomp=y|n
dcextendfunction=facility-extension-level-of-RPC-service
dccltinquiretime=maximum-time-interval-in-permanent-connection
dccltport=client-extended-service-port-number
dccltdccmhost=DCCM3-logical-terminal-host-name
dccltdccmport=DCCM3-logical-terminal-port-number
dccltxatmi=Y|N
dcwatchtiminherit=Y|N
dccltdelay=maximum-communication-delay-time
dccltcupsndhost=CUP-send-host
dccltcuprcvport=port-number-used-for-CUP-reception
dccltraphost=RAP-processing-listener or DCCM3-logical-terminal
dccltrapautoconnect=Y|N
dcclttrstatisitem=statistics-item
dcclttroptiitem=transaction-optimization-item
dcclttrwatchtime=maximum-communication-wait-time-during-transaction-sync
hronous-point-processing
dcclttrrbinfo=no|self|remote|all
dcclttrlimittime=maximum-transaction-execution-time
dcclttrrbrcv=Y|N
dcclttrrecoverytype=type1|type2|type3
dcwatchtimrpcinherit=Y|N
dcsyswatchtim=maximum-OpenTP1-control-response-wait-time
dccltauthent=y|n
dccltconnectinf=terminal-identification-information
dcscdmulti=Y|N
```

```
dcscdmulticount=number-of-multi-scheduler-daemon-processes
dchostselect=Y|N
dcscdloadpriority=Y|N
dccltonlythisnode=Y|N
dccltnoserver=Y|N
dcselint=reception-check-interval
dchostchange=Y|N
dccltoption=client's-extension-option
dccltnamextend=0|1
dctrcpath=trace-output-directory
dctrcerr=error-log-size
dctrcuap=UAP-trace-file-size
dctrcsoc=socket-trace-file-size
dctrcsocsize=socket-trace-record-size
dctrcmdl=module-trace-file-size
dccltprfinfosend=Y|N
dccltrecvbufsize=TCP/IP-receive-buffer-size
dccltsendbufsize=TCP/IP-send-buffer-size
dcclttcpnodelay=Y|N
```

#

This operand specifies TP1/Client/P definition start declaration and is mandatory. [ ] does not mean "omissible."

This operand can be omitted.

Include the above client environment definitions in the `betran.ini` file under the `Windows` directory.

When you use the `_s` version of a function (`dc`*_xxx_xxx*`_s`), you can use any client environment definition file you created. When you use your own client environment definition file, specify the pathname of the file in the `defpath` argument of the `dc_clt_cltin_s` function. The pathname must be specified with the absolute path or relative path from the current drive (current directory).

When you use the facility for receiving one-way messages from the server, you can specify any client environment definition file in the `dc_clt_accept_notification_s`, `dc_clt_cancel_notification_s`, or `dc_clt_open_notification_s` function.

7.2.3 TP1/Client/W

Defines the environment for using the OpenTP1 client function.

As a general rule, enter these definition commands in `/etc/profile` or `$HOME/.profile` for sh (Bourne shell), or in `/etc/cshrc` or `$HOME/.cshrc` for `csh` (C shell).

In a multi-thread environment, you can create your own client environment definition file that contains definitions in the TP1/Client/P format. To use your own client environment definition file, specify the absolute pathname of it in the `defpath` argument of the `dc_clt_cltin_s` function. If you want to use a different client environment definition for each thread, issue the `dc_clt_cltin_s` function, specifying the pathname of the desired file in the `defpath` argument for each thread.

7.2.4 TP1/Client/P

Defines the environment for using the OpenTP1 client function.

Create a client environment definition file named `BETRAN.INI` under the Windows directory (generally, `\WINDOWS`) and define the definition commands listed above in that file.

When you use the `_s` version of a function (`dc_`*xxx*`_`*xxx*`_s`), you can use any client environment definition file you created. When you use your own client environment definition file, specify the pathname of the file in the `defpath` argument of the `dc_clt_cltin_s` function. The pathname must be specified with the absolute path or relative path from the current drive (current directory).

When you use the facility for receiving one-way messages from the server, you can specify any client environment definition file in the `dc_clt_accept_notification_s`, `dc_clt_cancel_notification_s`, or `dc_clt_open_notification_s` function.

7.2.5 Operands common to TP1/Client/W and TP1/Client/P

For Windows, use lowercase alphabetic characters to specify operands.

For details about the operands that are specific to TP1/Client/W, see *TP1/Client/W-specific operands*. For details about the operands that are specific to TP1/Client/P, see *TP1/Client/P-specific operands*.

- `DCNAMPORT=`*name-service-port-number* ~<unsigned integer> ((5001-65535)) <<10000>>

  Specify the port number of the name service. The target TP1/Server must use this port number to start the name service.

  You can also specify the port number of the name service using `DCHOST` in the client environment definition.

- `DCHOST=`*TP1/Server-as-a-gateway* ~<character string>

  Specify the host computer name and the port number for the TP1/Server used as a gateway. You can specify two or more TP1/Servers as gateways using a comma (,) as a separator.

  Format:

host-computer-name[:*port-number*][,*host-computer-name*[:*port-number*],...]

host-computer-name~<character string>

port-number~<unsigned integer>((5001-65535))

You can specify a maximum of 63 characters for the host name. When `00000008` is specified for `DCCLTOPTION` in the client environment definition, you can specify a maximum of 255 characters. The maximum number of characters you can specify in this operand is 1,023.

Do not place a blank character (space or tab) except after the separator (,).

You can specify an IP address in decimal dot notation for the host name.

When the port number is not specified, the system uses the value for `DCNAMPORT` in the client environment definition.

When you have specified more than one TP1/Server in `DCHOST` and an error is detected in the TP1/Server being used as a gateway, system operation depends on the specification of `DCHOSTSELECT` in the client environment definition. If `N` is specified for `DCHOSTSELECT`, the system attempts to replace the failed node by referencing the next TP1/Server of the currently used TP1/Server. If `Y` is specified for `DCHOSTSELECT`, the system selects a TP1/Server at random (excluding the TP1/Server in which the error was detected) and attempts to replace the failed node.

Note that when you issue a client user authentication request with the gateway TP1/Server specified, this specification overrides the specification of `DCHOST` in the client environment definition. If the gateway TP1/Server is not specified either in a client user authentication request or in `DCHOST` in the client environment definition, TP1/Client uses a broadcast to determine the TP1/Server to be used as the gateway. To perform a broadcast in TP1/Client/P, you must specify the broadcast address in the `hosts` file (the host name must be `broadcast`).

- `DCWATCHTIM=`*maximum-response-wait-time*~<unsigned integer> ((0-65535)) <<180>> (unit: seconds)

  For response type RPCs, specify the maximum wait time between a CUP sending a service request to an SPP and the return of the service response. An error is returned to the CUP if no response is received within the specified duration.

  If `DCWATCHTIM` is set to zero, the function waits for a response indefinitely.

- `DCCLTCONNECTTIMEOUT=`*timeout-for-establishing-connection* ~<unsigned integer>((0-65535))<<0>>(unit: seconds)

  Specify the timeout for establishing connection in non-blocking mode for data transmission.

When you specify `0` or omit this operand, connection is established in blocking mode and the operating system monitors whether connection is established.

If this operand is incorrectly specified, the KFCA02401-E message is output to the error log and the `dc_clt_cltin_s` function returns a `DCCLTER_FATAL` error.

The function issued by the CUP may return an error before the time specified in this operand elapses. This error occurs, for example, when the remote system is not running. This error occurs because the operating system's timeout for establishing a connection prevails over the timeout specified in this operand. The operating system's timeout for establishing a connection differs depending on the platform you are using.

This operand monitors the time elapses after a connection establishment function of TP1/Client is executed. This time does not include the processing time of the function itself. Depending on the function or program you use, its processing time may be longer than the time specified in this operand.

- `DCCLTTREXPTM=`*expiry-time-in-transaction-branch* ~<unsigned integer> ((0-65535)) (Unit: seconds)

  Specify the maximum expiry time in a transaction branch. This operand is valid only when the transaction is started from a CUP.

  If the transaction branch is not completed within the specified expiry time, its process is terminated abnormally and rolled back. Specify `0` not to perform monitoring.

  If this operand is not specified, the process follows the specification of the `trn_expiration_time` operand in the client service definition. If the connection destination is a RAP-processing server, the process follows the specification of the `trn_expiration_time` operand in the RAP-processing listener service definition.

  When using the RPC function, use `DCCLTTREXPSP` to specify whether the processing time for transaction branches executed in other processes is to be included in the monitoring time.

- `DCCLTTREXPSP=Y|N|F`

  Specify whether the following time is to be included in the monitoring time for a transaction branch:

  The time required for a transaction branch of the transactional RPC executing process to call another transaction branch using the RPC function and to wait for completion of the processing

  `Y`: Includes the time in the monitoring time.

  `N` or `F`: Does not include the time in the monitoring time.

If this operand is not specified, the process follows the specification of the `trn_expiration_time_suspend` operand in the client service definition. If the connection destination is a RAP-processing server, the process follows the specification of the `trn_expiration_time_suspend` operand in the RAP-processing listener service definition.

- `DCCLTTRWATTM=`*maximum-time-interval-in-transaction-inquiry-response*~ <unsigned integer> ((1-65535))<<180>>(Unit: seconds)

  Specify the maximum time interval in transaction processing between an inquiry made from the CUP to a server (by issuing a transaction control function or the `dc_rpc_call_s` function) and another inquiry. This operand is valid only when a transaction is started from the CUP.

  The maximum time interval in transaction inquiry response must be smaller than the expiry time in transaction branch.

  If no inquiry occurs within the specified time, the system rolls back the transaction process on the server.

- `DCCLTTRCPUTM=`*CPU-monitoring-time-in-transaction-branch* ~<unsigned integer> ((0-65535))(Unit: seconds)

  Specify the CPU time that can be used by a transaction branch before synchronous-point processing. This operand is valid only when the transaction is started from a CUP.

  If `0` is specified, the CPU time is not monitored.

  If the specified time is exceeded, the process of the transaction branch is terminated abnormally and rolled back.

  If this operand is not specified, the process follows the specification of the `trn_cpu_time` operand in the client service definition. If the connection destination is a RAP-processing server, the process follows the specification of the `trn_cpu_time` operand in the RAP-processing listener service definition.

- `DCCLTUTTRCMT=Y|N` ~<<`N`>>

  Specify whether the transaction started from a CUP using the online tester is to be committed or rolled back.

  `Y`: The transaction is committed.

  `N`: The transaction is rolled back.

- `DCRCVPORT=`*Receive-CUP-port-number* ~<unsigned integer>((1-65535)) <<11000>>

  Specify the port number of the CUP that receives messages when you use the TCP/IP communication facility to receive messages. Specify this port number on the message-originating side. Specify a unique port number for each process or thread when multiple processes or multiple threads are executed simultaneously on the same machine.

Do not specify a port number for use by the operating system or other programs even if one can be specified. If you specify a port number in this case, response data might not be received correctly. The port numbers used by the operating system differ depending on the operating system. For details, see the documentation of your operating system.

- DCSNDHOST=*connected-node-name* ~<character string>

  Specify the host name of the node to be connected by establishing a connection when you use the TCP/IP communication facility to send messages.

  You can specify an IP address in decimal dot notation as a host name.

  You can specify a maximum of 63 characters for the host name. You can specify a maximum of 255 characters when 00000008 is specified for DCCLTOPTION in the client environment definition.

- DCSNDPORT=*connected-port-number* ~<unsigned integer>((1-65535)) <<12000>>

  Specify the port number of the node to be connected by establishing a connection when you use the TCP/IP communication facility to send messages.

- DCSOCKOPENATRCV=Y|N ~<<N>>

  Specify when to open the receive socket for sending and reception performed with one connection, when using the TCP/IP communication facility. Here, *when to open the receive socket* means when TP1/Client begins to wait for connection from the other party.

  This definition is effective only when DCCLT_SNDRCV is specified in the flags argument of the dc_rpc_open_s function.

  Y: Opens the receive socket if no connection has been established when any of the following functions is issued:

  - dc_clt_receive_s
  - dc_clt_receive2_s
  - dc_clt_assem_receive_s

  N: Opens the receive socket when the dc_rpc_open_s function is executed. This is the default.

- DCCLTDELIVERYCHECK=Y|N ~<<N>>

  Specify whether to use the message delivery confirmation facility.

  Y: The message delivery confirmation facility is used.

N: The message delivery confirmation facility is not used.

When the message delivery confirmation facility is used, if the dc_clt_assem_send_s function is issued, the function sends a message, receives response-only data, and then returns control. If the dc_clt_assem_receive_s function is issued, the function receives a message, sends response-only data, and then returns control.

If either the dc_clt_assem_send_s or dc_clt_assem_receive_s function is issued when this operand is set to N, the function uses the message assembly facility to send or receive a message.

- DCUTOKEY=*test-user-ID* ~<1-4 alphanumeric characters>

Specify this operand when executing a CUP using the online tester.

With this operand specified, SPPs started from the CUP can be executed in test mode.

- DCCACHE=*number-of-areas-for-temporarily-storing-service-information* ~<unsigned integer> ((2-10240)) <<8>>

For when you execute an RPC from a client, specify the number of cache areas that store the service information provided by the TP1/Server name service that is used as a gateway. Each cache area stores one service information item.

The stored service information is deleted from the cache areas when the effective period, specified using DCCLTCACHETIM in the client environment definition, expires.

When you specify this operand, use the following as a guideline:

When DCCLTLOADBALANCE=N is specified

Specify the number of servers to which the client sends RPC-based requests.

When DCCLTLOADBALANCE=Y is specified

Specify the number of all servers in the nodes that are started as multi-node servers.

Each cache area consumes about 150 bytes of memory. When DCSCDDIRECT=Y is specified in the client environment definition, this operand is invalid.

- DCCLTCACHETIM=*expiration-of-temporarily-stored-service-information* ~<unsigned integer> ((0-65535)) <<30>> (Unit: seconds)

Specify the effective period for the service information that is acquired from the TP1/Server name service that is used as a gateway. When the effective period expires, the service information is deleted from the cache areas. When you specify 0, no effective period is specified. Once the service information is stored, it will be valid until the dc_rpc_close_s function is issued or the existing information is overwritten by new service information when free cache areas run short. This operand is valid only when `DCCLTLOADBALANCE=Y` is specified in the client environment definition. This operand is invalid when `DCSCDDIRECT=Y` is specified in the client environment definition.

- `DCCLTLOADBALANCE=Y|N` ~<<N>>

Specify whether to use the inter-node load-balancing facility when multi-node servers are used. The inter-node load-balancing facility evaluates the load status of each node internally within TP1/Client when an RPC is executed, and distributes the load to servers with less load.

`Y`: Uses the inter-node load-balancing facility.

`N`: Does not use the inter-node load-balancing facility.

When you specify `Y` for this operand, specify the following operands in the client environment definition according to the number of nodes or the number of servers that handle RPC-based requests.

- `DCCACHE`
- `DCCLTCACHETIM`
- `DCCLTNAMEXTEND`

This operand is invalid when `DCSCDDIRECT=Y` is specified in the client environment definition.

- `DCCLTSERVICEGROUPLIST=`*file-defining-correspondence-between-service-groups-and-RPCs* ~<character string>

Specify a text file (with its path name) that defines correspondence between the service group and the RPC entry point for the corresponding server. This file is used for issuing an RPC to a server other than OpenTP1. Define this file as follows.

Format:

<*service-group-name*> <*server-host-computer-name*>
<*entry-point-port-number*> `[,`<*server-host-computer-name*>
<*entry-point-port-number*>`,...] [`<*comment*>`]`

- Define a pair of the service group and the RPC entry point per line in a text file. When two or more RPC entry points are available, use as many lines as these entry points.
- Separate each item with a space or a tab.
- Specify each item as follows.

service-group-name

Any character string with up to 31 characters.

server-host-computer-name

Host name of the connection destination host. You can specify a maximum of 63 characters for the host name. You can specify a maximum of 255 characters when `00000008` is specified for `DCCLTOPTION` in the client environment definition. You can also specify an IP address in decimal dot notation as a host name.

entry-point-port-number

Numeric value between 1 and 65535 as a port number for accepting RPCs.

comment

Any text beginning with `#`. This is assumed to be a comment to the end of the line and is ignored for the processing. The comment is optional.

When you specify multiple RPC entry points, the system selects one of them at random and attempts to connect it. If an attempt to connect the selected host fails, that host is eliminated from the connection destination options. Then, the system selects another RPC entry point at random in an attempt to connect. This step will be repeated. If all the attempts to connect a RPC entry point fail, the `dc_rpc_call_s` function returns an error.

Evaluating the file content:

When the file is defined incorrectly, TP1/Client ignores the invalid line. It also outputs the corresponding line number to the error log.

If the name of the service group called by the `dc_rpc_call_s` function has not been defined in the file specified in this operand, operation follows the specification of the `DCCLTNOSERVER` operand in the client environment definition:

- When `DCCLTNOSERVER=Y` is specified:

  The function immediately returns a `DCRPCER_NO_SUCH_SERVICE_GROUP` error.

- When `DCCLTNOSERVER=N` is specified or the `DCCLTNOSERVER` operand is omitted:

  On recognizing that the name of the called service group is not defined in the file specified in this operand, the function performs an RPC to TP1/Server.

■ `DCCLTCONNECTRETRY=`*retry-count-for-connection-establishment* ~<unsigned integer> ((0-255)) <<0>>

Specify the maximum number of times a request to establish connection should be retried if a timeout occurred for a request because the server was offline or turned off, for example. When `0` is specified in this operand or when this operand is not specified, TP1/Client does not perform a retry. If this operand is specified incorrectly, TP1/Client assumes `0`. This definition is effective when you attempt user authentication (specify `DCCLTAUTHENT=Y` and execute the `dc_clt_cltin_s` function with `DCNOFLAGS` specified in the `flags` argument).

- `DCSCDDIRECT=Y|N` ~<<N>>

  Specify whether to use the function to directly inquire into the schedule service without inquiring service information from the TP1/Server name service (RPC that does not use the name service).

  `Y`: Uses an RPC that does not use the name service.

  `N`: Does not use an RPC that does not use the name service.

  When `DCSCDPORT` in the client environment definition specifies the port number for the schedule service, the client makes an inquiry using that port number. When no `DCSCDPORT` is defined, the client acquires the port number for the schedule service from TP1/Server, then makes an inquiry.

  When this function is used, the program cannot call socket-receiving type SPPs. Definitions of `DCCACHE`, `DCCLTCACHETIM`, and `DCCLTLOADBALANCE` in the client environment definition are ignored. When `DCCLTSERVICEGROUPLIST` is specified in the client environment definition, the definition of `DCSCDDIRECT` is ignored.

- `DCSCDPORT=`*schedule-service-port-number* ~<unsigned integer> ((5001-65535))

  Specify the port number of the schedule service. When `DCSCDMULTI=Y` and `DCSCDDIRECT=Y` are specified in the client environment definition, specify the port number which is used as the base of the multi-scheduler daemon. The target TP1/Server must use this port number to start the schedule service or the multi-scheduler daemon. For details about the specification of the schedule service or the multi-scheduler daemon of TP1/Server, see the manual *OpenTP1 System Definition*.

  This operand is valid only when `DCSCDDIRECT=Y` is specified in the client environment definition.

  When `DCSCDMULTI=Y` and `DCSCDDIRECT=Y` are specified in the client environment definition, also see the description of `DCSCDMULTICOUNT` in the client environment definition.

  If you omit this operand, the name service is sent a request regarding the port number of the schedule service or the multi-scheduler daemon.

- `DCCLTDATACOMP=Y|N` ~<<N>>

Specify whether to use the data compression.

`Y`: The data compression is used.

`N`: The data compression is not used.

- `DCEXTENDFUNCTION`=*facility-extension-level-of-RPC-service* ~<unsigned hexadecimal integer> ((00000000-00000001)) <<00000000>>

  Specify either of the following as the extension level of the RPC service facility. Specify all the zeros written below. Even if you specify a value that cannot be specified, TP1/Client may operate incorrectly without generating an error code.

  `00000000`

  The RPC service facility is not extended.

  `00000001`

  If the SPP that is executing a service request terminates abnormally, the `dc_rpc_call_s` function return an error code (`DCRPCER_SERVICE_TERMINATED`) to isolate the error. If this specification is not made, `DCRPCER_TIMED_OUT` and `DCRPCER_SERVICE_NOT_UP` are returned.

  This definition is invalid when a service request (`dc_rpc_call_s` function) is executed during establishment of a permanent connection or within the scope of a transaction. During establishment of a permanent connection or within a transaction, the specification of the `rpc_extend_function` operand in the user service default definition is valid.

- `DCCLTINQUIRETIME`=*maximum-time-interval-in-permanent-connection* <unsigned integer> ((0-1048575)) (unit: seconds)

  Specify the maximum interval between an inquiry from the CUP to the server and the next inquiry. The CUP execution process or RAP-processing-server monitors this interval, and forcibly releases the permanent connection if no inquiry is made within the specified period of time.

  This definition is ineffective when you establish permanent connection with a DCCM3 logical terminal. DCCM3 provides the equivalent feature using the terminal monitoring time. For DCCM3, specify the terminal monitoring time in the LEFTLIMIT clause of the TERMINAL statement for the data communication definition.

  If expiration of the maximum interval is detected in a transaction, the transaction is rolled back.

  If `0` is specified, the system waits infinitely for an inquiry from the CUP. When you omit this specification, the process follows the specification of the `clt_inquire_time` operand in the client service definition or `rap_inquire_time` operand in the RAP-processing listener service definition. If the connection destination is a RAP-processing server, the process follows the specification of the `rap_inquire_time` operand in the RAP-processing listener service definition.

The maximum time interval in permanent connection specified in the definition monitors the time between the `dc_clt_connect_s` function and the `dc_clt_disconnect_s` function when a permanent connection is established. The maximum time interval in transaction inquiry response (`DCCLTTRWATTM`) monitors the time between inquiries up to the `dc_trn_unchained_commit_s` function when the `dc_trn_begin_s` function is issued without establishing a permanent connection.

■ `DCCLTPORT=`*client-extended-service-port-number* ~ <unsigned integer> ((5001-65535))

Specify the port number of the client extended service. The target TP1/Server must use this port number to start the client extended service. Specify the port number of the client extended service using the `clt_port` operand in the client service definition.

If this definition is not specified, the port number of the client extended service is inquired from the name service.

■ `DCCLTDCCMHOST=`*DCCM3-logical-terminal-host-name*

When you request establishing the permanent connection with a DCCM3 logical terminal, specify the host name of the connection target logical terminal. At this time, specify `DCCLT_DCCM3` in the flags argument of the `dc_clt_connect_s` function.

Specify as follows.

host-name:[*port-number*][,*host-name*[:*port-number*],...]

> *host-name* ~<character string>
>
> *port-number* ~<unsigned integer> ((1-65535))
>
> You can specify a maximum of 63 characters for the host name. When `00000008` is specified for `DCCLTOPTION` in the client environment definition, you can specify a maximum of 255 characters. The maximum number of characters you can specify in this operand is 1,023.
>
> You can use a blank character (space or tab) only after a delimiter (,).

You can specify an IP address in decimal dot notation as a host name.

If you do not specify a port number, the port number of the DCCM3 logical terminal specified in the `DCCLTDCCMPORT` client environment definition is assumed.

When you specify multiple DCCM3 logical terminals, the system selects one of them at random and attempts to connect it. If an attempt to connect to the selected DCCM3 logical terminal fails, that DCCM3 logical terminal is eliminated from the connection destination options. Then, the system selects another DCCM3 logical terminal at random in an attempt to connect. This step will be repeated. If all the attempts to connect a DCCM3 logical terminal fail, the `dc_clt_connect_s` function returns an error.

When you establish a permanent connection to communicate with a DCCM3 logical terminal, the value specified in the `DCCLTSERVICEGROUPLIST` client environment definition is ignored. Note that the data compression cannot be performed.

- `DCCLTDCCMPORT=`*DCCM3-logical-terminal-port-number* ~ <unsigned integer> ((1-65535)) <<30000>>

  Specify the port number with which the CUP makes a request of permanent connection to the DCCM3 logical terminal.

- `DCCLTXATMI=Y|N` ~ <<N>>

  Specify whether to use the XATMI interface for communication. If this operand is specified incorrectly, TP1/Client assumes `N`.

  `Y`: Use the XATMI interface.

  `N`: Do not use the XATMI interface.

- `DCWATCHTIMINHERIT=Y|N` ~ <<N>>

  Specify whether an extended client service should inherit the maximum CUP response wait time for transaction or connection control.

  `Y`: An extended client service inherits the maximum CUP response wait time.

  `N`: An extended client service does not inherit the maximum CUP response wait time.

  When you specify `Y` for this operand, refer to `DCCLTDELAY` in the client environment definition.

- `DCCLTDELAY=`*maximum-communication-delay-time* ~ <unsigned integer> ((0-65535)) <<0>> (units: seconds)

  Specify this item when terminating server response monitoring earlier than the client response monitoring, considering the communication overhead between the CUP and an extended client service. This definition allows terminating the server monitoring quickly by the specified time. Prompt termination prevents the improper transmission of messages due to the time-out of client monitoring.

  This definition is valid only when `Y` is set for `DCWATCHTIMINHERIT` in the client environment definition. The definition of `DCCLTDELAY` is ignored when `0` is set for client environment definition DCWATCHTIM. If the subtraction of a value specified in DCCLTDELAY from a value specified in DCWATCHTIM provides 0 or negative value, the definition of DCCLTDELAY is ignored, with 1 defaulted to.

You can dynamically change a value specified for DCWATCHTIM in the client environment definition by issuing the dc_rpc_set_watch_time_s function. If such value has been dynamically changed, the new value is calculated by the dc_rpc_watch_time_s function.

- DCCLTCUPSNDHOST=*CUP-send-host* ~<character string>

Specify the host that sends a connection establishment request.

You can specify a maximum of 63 characters for the host name. If you specify 00000008 for DCCLTOPTION in the client environment definition, you can specify a maximum of 255 characters for the host name.

You can also specify an IP address in decimal dot notation for the host name.

If you specify localhost as the host name or an IP address that begins with 127, the dc_clt_cltin_s function returns a DCCLTER_FATAL error.

If you specify a host that is not on the same machine on which the CUP runs, the communication function returns a DCCLTER_NET_DOWN or DCRPCER_NET_DOWN error.

If you omit this operand, the send host is assigned automatically.

- DCCLTCUPRCVPORT=*port-number-used-for-CUP-reception* ~ <unsigned integer> ((5001-65535))

Specify the port number of a CUP that receives a message from the server.

The port number specified in this definition is valid when you use the following functions.

- Ordinary RPC function (for reception)
- Transaction control function
- Permanent connection establishment function

If you omit this operand, the system uses the port number assigned by it.

Specify a unique port number for each process or thread when multiple processes or multiple threads are executed simultaneously on the same machine.

Do not specify a port number for use by the operating system or other programs even if the port number is valid. If you specify a port number in this case, response data might not be received correctly. The port numbers used by the operating system differ depending on the operating system. For details, see the documentation of your operating system.

- DCCLTRAPHOST=*RAP-processing-listener* or *DCCM3-logical-terminal*

  Specify the host name and port number of a RAP-processing listener or the DCCM3-logical-terminal supported in TP1/Server.

  The following items give the formats.

  host-name:*port-number*[,*host-name*:*port-number*,...]

  host-name ~ <character string>

  port-number ~ <unsigned integer> ((5001-65535))

  You can specify a maximum of 63 characters for the host name. When 00000008 is specified for DCCLTOPTION in the client environment definition, you can specify a maximum of 255 characters. The maximum number of characters you can specify in this operand is 1,023.

  You can use a blank character (space or tab) only after a delimiter (,).

  You can also specify an IP address in decimal dot notation as a host name.

  If you issue the dc_clt_connect_s function with DCNOFLAGS set at flags, the system requests a RAP-processing listener or the DCCM3 logical terminal for TP1/Server to establish a permanent connection when you specify this definition. If you do not specify this definition, the system requests an extended client service for TP1/Server to establish a permanent connection.

  When you request establishing the permanent connection with a DCCM3 logical terminal, the data compression is unavailable.

  Note, however, that, if a firewall exists between the CUP and the RAP-processing listener or the DCCM3 logical terminal, the host name and port number specified in DCCLTRAPHOST become the host name and port number of the firewall, respectively.

  When you specify several pairs of a host name and the port number of the RAP-processing listener (or the port number of the DCCM3 logical terminal), the system selects one of the pairs at random and attempts to connect the selected host. If an attempt to connect the selected host fails, that host is eliminated from the connection destination options. Then, the system selects another host at random in an attempt to connect. This step will be repeated. If all the attempts to connect a host fail, the dc_clt_connect_s function returns an error.

  When you establish a permanent connection to communicate with a RAP-processing listener or with a DCCM3 logical terminal, the value specified in the DCCLTSERVICEGROUPLIST client environment definition is ignored.

  The following table shows the relationships between the client environment definition and flags on the dc_clt_connect_s function. You can find the target of permanent connection.

Argument flags	Client environment definition		Establishing permanent connection for:
	DCCLTDCCMHOST	DCCLTRAPHOST	
`DCNOFLAGS`	Y	Y	RAP-processing server or DCCM3 logical terminal[1]
			CUP execution process
		Y	RAP-processing server or DCCM3 logical terminal[1]
			CUP execution process
`DCCLT_DCCM3`	Y	Y	DCCM3 logical terminal[2]
			DCCM3 logical terminal[2]
		Y	Error return
			Error return

Legend:

Y: Specified

Blank: Not specified

1

Establish permanent connection for a DCCM3 logical terminal specified by `DCCLTRAPHOST`.

2

Establish permanent connection for a DCCM3 logical terminal specified by `DCCLTDCCMHOST`.

- `DCCLTRAPAUTOCONNECT=Y|N` ~ <<N>>

Specify whether to automatically establish permanent connection between a CUP and a RAP-processing server or between a CUP and a DCCM3 logical terminal.

`Y`: Automatically establishes a permanent connection.

`N`: Does not automatically establish a permanent connection.

Specifying `Y` in this definition automatically establishes the permanent connection if the permanent connection is not yet established when the following functions are executed. The requested permanent connection destination is a RAP-processing listener or a DCCM3 logical terminal defined in the `DCCLTRAPHOST` client environment definition.

1. `dc_rpc_call_s` function
2. `dc_trn_begin_s` function

However, if you execute function number 2 for a DCCM3 logical terminal, an error is returned.

When you specify `Y` in this definition, you need not execute the `dc_clt_connect_s` function. In addition, the `dc_clt_disconnect_s` function need not be executed since the permanent connection is automatically released when the `dc_rpc_close_s` function is executed.

■ `DCCLTTRSTATISITEM=`*statistics-item*`[,`*statistics-item*`]...`

Specify the string constant that specifies the transaction branch statistics to be acquired. This definition is valid only when a transaction is started from the CUP.

`nothing`

Does not acquire statistics.

`base`

Acquires the following information as basic information.

Transaction branch's identifier

Result of the transaction branch's settlement

Type of the transaction branch's execution process

Name of the transaction branch's execution server

Name of the transaction branch's execution service

`executiontime`

Acquires basic information and execution time information about a transaction branch.

`cputime`

Acquires basic information and CPU time information about a transaction branch.

You can specify `nothing` only once. Another statistics item has priority over nothing.

When acquiring statistics about a transaction, specify one of the following.

- `trn_tran_statistics=Y` in the transaction service definition
- `-s` option in the `trnstics` command

If this operand is not specified, the system follows the specification of the `trn_statistics_item` operand in the client service definition. If the connection destination is a RAP-processing server, the process follows the specification of the `trn_statistics_item` operand in the RAP-processing listener service definition.

■ `DCCLTTROPTIITEM=`*transaction-optimization-item*`[,`*transaction-optimization-item*`]...`

Specify, by the following character string, an optimization item for enhancing the performance of a global transaction consisting of two or more user servers. This definition is valid only when a transaction is started from the CUP.

`base`

Optimizes the entire synchronous point acquisition processing (preparation, commit and rollback processing). OpenTP1 transaction control is performed in a two-phase commit way. Commit control between two transaction branches, therefore, requires four cycles of inter-process communication.

If all of the following requirements are satisfied, the parent transaction branch performs commit processing instead of child transaction branches to reduce four cycles of inter-process communication required for commit control.

- The parent transaction branch and child transaction branches are located under same OpenTP1.

- The parent transaction branch uses a synchronous response RPC to call child transaction branches.

- The object for the XA interface for a resource manager accessed by a child transaction branch is also linked to the parent transaction branch.

`asyncprepare`

Optimizes prepare processing if the system cannot optimize all the synchronous point acquisition processing because requirements on base are not satisfied. If all of the following requirements are satisfied, the system performs preparation before an RPC return when a child transaction branch executes a service request using an RPC issued from the parent transaction branch. The result is that two cycles of inter-process communication are reduced.

Processing cannot be optimized using base. The parent transaction branch uses a synchronous response type PRC to call a child transaction.

This optimization, however, delays the response time of a synchronous response type of RPC issued by the parent transaction branch. For a child transaction branch, this optimization increases the interval from prepare processing to commit processing. (In this status, a transaction cannot be settled if directions are not given from the parent transaction branch.) If

OpenTP1 for the parent transaction branch is out of order, the communication between transaction branches impossible. Therefore, that lack of communication between those branches delays the validation of a journal file swap and check point dump file. The result may be that OpenTP1 for a child transaction branch is also out of order.

A duplicate transaction optimization item can be specified. Base, however, has priority over `asyncprepare`.

If this operand is not specified, the system follows the specification of the `trn_optimum_item` operand in the client service definition. If the connection destination is a RAP-processing server, the process follows the specification of the `trn_optimum_item` operand in the RAP-processing listener service definition.

- `DCCLTTRWATCHTIME=`*maximum-communication-wait-time-during-transaction-synchronous-point-processing* ~ <unsigned integer> ((1-65535)) (units: seconds)

  Specify the maximum reception wait time for the communication between transaction branches (prepare, commit, rollback directions or response) during transaction synchronous point processing. This definition is valid only when a transaction is started from the CUP.

  If the system does not give directions or response within a specified period, the transaction branch is rolled back if the first phase of two-phase commit is being processed. After the completion of the first phase, the system retries transaction settlement processing in a system process for a transaction service.

  If this operand is not specified, the system follows the specification of the `trn_watch_time` operand in the client service definition. If the connection destination is a RAP-processing server, the process follows the specification of the `trn_watch_time` operand in the RAP-processing listener service definition.

- `DCCLTTRRBINFO=no|self|remote|all`

  Specify whether to save information about the cause of a rollback as logs if a transaction branch has been rolled back. This definition is valid only when a transaction is started from the CUP.

  `no`

  Does not obtain rollback information.

  `self`

  Obtains rollback information in a log. The rollback information is only for the transaction branch that caused the rollback.

  `remote`

  Obtains rollback information in a log. The rollback information is the same information as `self`, plus the rollback information for any transaction branch that requested a rollback from another node's transaction branch.

`all`

Obtains rollback information in a log. The rollback information is the same information as `remote`, plus the rollback information for any transaction branch that requested a rollback from the local node's transaction branch.

If this operand is not specified, the system follows the specification of the `trn_rollback_information_put` operand in the client service definition. If the connection destination is a RAP-processing server, the process follows the specification of the `trn_rollback_information_put` operand in the RAP-processing listener service definition.

■ `DCCLTTRLIMITTIME=`*maximum-transaction-branch-execution-time* ~ <unsigned integer> ((0-65535)) (units: seconds)

Specify the maximum transaction branch execution time. This definition is valid only when a transaction is started from the CUP.

The system automatically sets the expiration time of communication in synchronous point processing and of the `dc_rpc_call_s` function. This automatic setting prevents the period from the start of a transaction branch to the termination of synchronous point processing from exceeding a period specified in this operand.

- Time-out period of the `dc_rpc_call_s` function

  If K is equal to or greater than a period specified in this operand, the system does not perform request processing, and returns as if a timeout error occurred.

  If K is smaller than a period specified in this operand and if W is equal or less than a period specified in this operand minus K, the system adopts W as the time-out period.

  If K is less than a period specified in this operand and if W is greater than a period specified in this operand minus K, the system adopts the period specified in this operand minus K as the time-out period.

  K: Current time minus transaction branch start time

  W: Time specified in the `DCWATCHTIM` operand

- Time-out period of communication in synchronous point processing

  If K is equal to or greater than a period specified in this operand, the expiration time is one second.

  If K is less than a period specified in this operand and if W is equal or less than a period specified in this operand minus K, the system adopts W as the time-out period.

If K is less than a period specified in this operand and if W is greater than a specified period in this operand minus K, the system adopts the period specified in this operand minus K as the time-out period.

K: Current time minus transaction branch start time

W: Time specified in the `DCCLTTRWATCHTIME` operand (or, if the `DCCLTTRWATCHTIME` operand is omitted, the time specified in the `DCWATCHTIM` operand)

If processing other than the above reception wait takes much time, a transaction branch may not be terminated within a period specified in this operand.

If the period specified in this operand has passed before the start of synchronous point processing, the transaction is rolled back.

If you specify `0`, the system does not monitor the time.

If this operand is not specified, the system follows the specification of the `trn_limit_time` operand in the client service definition. If the connection destination is a RAP-processing server, the process follows the specification of the `trn_limit_time` operand in the RAP-processing listener service definition.

- `DCCLTTRRBRCV=Y|N`

  Specify whether to receive notice of the completion of a rollback after sending rollback directions to a destination transaction branch of an RPC. This definition is valid only when you start a transaction from the CUP.

  `Y`: Receives rollback completion notification.

  `N`: Does not receive rollback completion notification.

  If you specify `N`, the system terminates the local transaction branch without receiving notice of the completion of a rollback from a destination transaction branch of an RPC (without waiting for the completion of rollback processing in a destination transaction branch of an RPC).

  If this operand is not specified, the system follows the specification of the `trn_rollback_response_receive` operand in the client service definition. If the connection destination is a RAP-processing server, the process follows the specification of the `trn_rollback_response_receive` operand in the RAP-processing listener service definition.

- `DCCLTTRRECOVERYTYPE=type1|type2|type3`

  Specify a method for processing a transaction synchronous point when a UAP incurs an error. This definition is valid only when a transaction is started from the CUP.

If the time-out of an RPC occurs and the address of a process for an RPC destination is not determined or if a UAP for executing a transaction gets out of order, a transaction branch may not smoothly communicate with another transaction. In this case, therefore, transaction settlement may take a long time.

If the following failures occur, the system selects one of the following three methods of transaction synchronous point processing.

`type1`

Failure 1: Time-out of an RPC

In this case, the RPC source transaction branch cannot determine the process in which a service request is being executed. It cannot, therefore, send a transaction synchronous point message to the RPC source transaction branch. Both of the RPC source transaction branch and the RPC destination transaction branch wait for a transaction synchronous point message. The result is that transaction settlement takes a long time.

`type2`

Failure 2: Malfunction of the RPC source UAP before reception of an RPC response

In this case, the RPC source transaction branch cannot determine the process executing a service. It cannot, therefore, send a transaction synchronous point message to the RPC source transaction branch. The RPC destination transaction branch waits for a transaction synchronous point message. The result is that transaction settlement takes a long time.

`type3`

Failure 3: Nearly concurrent malfunction of the RPC source and destination UAPs after reception of a response from the RPC source UAP

In this case, the transaction recovery process that has inherited each transaction branch is not notified that the remote UAP process is out of order. It, therefore, sends a transaction synchronous point message to a UAP process that does not exist. The result is that transaction settlement may take a long time.

In the following cases, transaction settlement may take a long time even if this operand is assigned `type2` or `type3`.

- During the execution of an RPC, the status of the RPC destination UAP was changed (because of an increase in the load, the termination of the UAP, shutdown or the like). A service request has been re-transferred to the same UAP in another node.
- The version of destination OpenTP1 does not support this option.

- The destination transaction branch is occupied by processing other than processing for transaction synchronous point message reception.

If this operand is not specified, the system follows the specification of the `trn_partial_recovery_type` operand in the client service definition. If the connection destination is a RAP-processing server, the process follows the specification of the `trn_partial_recovery_type` operand in the RAP-processing listener service definition.

■ `DCWATCHTIMRPCINHERIT=Y|N` ~ <<N>>

Specify whether the server should inherit the maximum wait time for a CUP response. By inheriting the maximum wait time for a CUP response, the server can be prevented from executing a service if the time-out of the CUP occurs.

`Y`: The server inherits the maximum wait time for a CUP response.

`N`: The server does not inherit the maximum wait time for a CUP response.

■ `DCSYSWATCHTIM=`*maximum-OpenTP1-control-response-wait-time* ~ <unsigned integer> ((0-65535)) <<maximum-response-wait-time>> (units: seconds)

Specify the maximum value for the wait time from the sending of a request to the return of a response when under the control of OpenTP1. If a response is not returned within a specified period, the system returns an error message to the CUP.

If you specify `0`, the system waits endlessly for the return of a response. In the default mode, the system applies `DCWATCHTIM` to the client environment definition.

You cannot dynamically change the maximum wait time for the OpenTP1 control response.

■ `DCCLTAUTHENT=Y|N` ~ <<Y>>

Specify whether to authenticate the user for executing the `dc_clt_cltin_s` function.

`Y`: Authenticates the user.

`N`: Suppresses user authentication.

■ `DCCLTCONNECTINF=`*terminal-identification-information*

Specify the terminal identification information. If you want to use hexadecimal numbers to specify this information, add 0x at the beginning of the information and use up to 128 digits (excluding the beginning 0x). If you use a character string, you can specify up to 64 characters.

When you use a permanent connection to communicate with a DCCM3 logical terminal, use EBCDIK code to specify the logical terminal name of the DCCM3 logical terminal as the terminal identification information. However, DCCM3 only validates the first 8 bytes (the 9th and later bytes are ignored).

The terminal identification information specified in this definition is referenced by the `dc_rpc_open_s` function. These functions then report that information to the DCCM3 logical terminal.

If you omit this definition, terminal identification information is not reported to the DCCM3 logical terminal.

However, if you execute the `dc_clt_set_connect_inf_s` function, the terminal identification information specified in this function is reported to the DCCM3 logical terminal when the `dc_clt_connect_s` function is executed.

This definition is valid when the host name and the port number of the DCCM3 logical terminal are specified in the `DCCLTRAPHOST` client environment definition and the `dc_clt_connect_s` function (specify `DCNOFLAGS` in the flags argument) is executed.

- `DCSCDMULTI=Y|N` ~<<N>>

  Specify whether to use the multi-scheduler facility.

  `Y`: Uses the multi-scheduler facility.

  `N`: Does not use the multi-scheduler facility.

  When you use the multi-scheduler facility, the system randomly selects one of multiple multi-scheduler daemons that are activated, to reduce the scheduling load.

  When you specify `Y` for this operand, also see the description of `DCSCDDIRECT`, `DCSCDPORT`, and `DCSCDMULTICOUNT` in the client environment definition. This operand is invalid when the `dc_rpc_call_to_s` function is executed.

- `DCSCDMULTICOUNT=`*number-of-multi-scheduler-daemon-processes* ~<unsigned integer> ((1-4096)) <<1>>

  Specify the number of multi-scheduler daemon processes. Specify the number of processes specified in the `-m` option in the `scdmulti` schedule service definition or less.

  This operand is valid when `DCSCDMULTI=Y`, `DCSCDDIRECT=Y`, and `DCSCDPORT` are specified in the client environment definition. In this case, the system randomly selects one of the port numbers in the following range:

  A to (A + B - 1)

  where,

  A: The port number specified in `DCSCDPORT`

  B: The number of processes specified in `DCSCDMULTICOUNT`

- Lower limit: Port number value specified in DCSCDPORT of the client environment definition
- Upper limit: Lower limit value + the number of processes specified in DCSCDMULTICOUNT of the client environment definition - 1

■ DCHOSTSELECT=Y|N ~<<N>>

Specify whether you want to have a gateway TP1/Server selected at random. This definition is valid only when multiple TP1/Servers are assigned as gateways.

Y: Selects TP1/Server to be used as the gateway at random.

N: Does not select TP1/Server to be used as the gateway at random.

When you specify Y, a gateway TP1/Server is selected at random from TP1/Servers that are specified at user authentication.

If an error occurs while the system is inquiring information from the name service of the gateway TP1/Server, TP1/Server where the error occurred is eliminated from the options. Then, the system selects another gateway TP1/Server at random in an attempt to switch.

When you specify N, gateway TP1/Servers are selected sequentially from the beginning of TP1/Servers that are specified at user authentication or those specified in the DCHOST client environment definition. If an error occurs while the system is inquiring information from the name service of the gateway TP1/Server, the system attempts to switch to the next specified TP1/Server.

When you specify Y in the DCSCDDIRECT client environment definition, the system attempts to switch the TP1/Server gateway if sending to the applicable port number fails.

■ DCSCDLOADPRIORITY=Y|N ~<<N>>

Specify whether to distribute the load on the gateway TP1/Server that receives service requests with priority.

Y: Distributes the load on the gateway TP1/Server that receives service requests with priority.

N: Accepts the specification of the scd_this_node_first operand in the schedule service definition.

This definition is valid only when an RPC is executed without using the name service (specify Y in the DCSCDDIRECT client environment definition).

■ DCCLTONLYTHISNODE=Y|N ~<N>

Specify whether to execute services on the specified node when the dc_rpc_call_to_s function is issued.

Y: Executes services only on the specified node.

N: First, attempts to execute services on the specified node. Depending on the service execution status, service requests may be transferred to another node.

- DCCLTNOSERVER=Y|N ~<<N>>

  Specifies whether the environment being used is an environment in which TP1/Client never communicates with TP1/Server.

  Y: In the environment being used, TP1/Client communicates with only a DCCM3 logical terminal and never communicates with TP1/Server.

  N: In the environment being used, TP1/Client communicates with TP1/Server.

  Always issue the dc_clt_cltin_s function even when Y is specified. In the logname argument, specify a value other than NULL. If you specify NULL, the dc_clt_cltin_s function returns a DCCLTER_INVALID_ARGS error. In the passwd argument, specify any value you like. You can specify NULL in the passwd argument.

- DCHOSTCHANGE=Y|N ~<<Y>>

  Specify whether to switch the gateway TP1/Server to another TP1/Server if TP1/Client receives an error from the gateway TP1/Server in one of the following cases: (1) when a service request is issued (when the dc_rpc_call_s function is executed), (2) when the schedule service is being started, and (3) when the schedule service is being terminated.

  Y: Switches the gateway TP1/Server.

  N: Does not switch the gateway TP1/Server.

  If this operand is incorrectly specified, TP1/Client assumes that Y is specified.

  When N is specified, a service request (the dc_rpc_call_s function) immediately returns an error. If the schedule service is being started, a DCRPCER_OLTF_INITIALIZING error returns. If the schedule service is being ended, a DCRPCER_OLTF_NOT_UP error returns.

  This definition is valid when more than one TP1/Server is specified with the target_host argument of the user authentication function and is specified with DCHOST in the client environment definition. This definition is invalid when a service request (the dc_rpc_call_s function) is executed during establishment of a permanent connection or within a transaction.

- DCCLTOPTION=*client's-extension-option* ~<unsigned hexadecimal integer> ((00000000-00000008)) <<00000000>>

  Specify the option for extending the functionality of the client. To specify multiple options, specify the logical sum of the values of them. Even if you specify a value that cannot be specified, TP1/Client may operate incorrectly without generating an error code.

`00000000`

When this option is specified, the functionality is not extended.

`00000002`

When this option is specified, the client's timeout for awaiting a response (the value of `DCSYSWATCHTIM` or `DCWATCHTIM` in the client environment definition) is applied to reception processing of communication functions. This timeout is not applied every time a reception occurs. This timeout is applied only to the wait for a response from communication functions.

Note that processing may be delayed if, for example, a fraction occurs during the decrement of time.

`00000008`

When this option is specified, the host name length that TP1/Client can handle is extended from 63 characters to 255 characters.

If this operand is specified incorrectly, the KFCA02401-E message is output to the error log, and one of the following functions returns a `DCCLTER_FATAL` error.

- `dc_clt_cltin_s` function
- `dc_clt_accept_notification_s` function
- `dc_clt_cancel_notification_s` function
- `dc_clt_open_notification_s` function

■ `DCCLTNAMEXTEND=0|1` ~<<0>>

Increase the number of service information items acquired and managed by clients when multi-node servers are used.

`0`: Acquires a maximum of 128 service information items.

`1`: Acquires a maximum of 512 service information items.

Specify `1` for this operand when `nam_service_extend=1` is specified in the name service definition of the TP1/Server that is used as gateway in a multi-node server configuration with 129 or more servers.

This operand is valid only when `DCCLTLOADBALANCE=Y` is specified in the client environment definition. When `DCSCDDIRECT=Y` is specified in the client environment definition, the specification of this operand is invalid.

■ `DCTRCPATH=`*trace-output-directory* ~<character string>

Specify the absolute pathname of the directory for storing the error log files and trace files.

No files are output if the specified directory does not exist or the specification is invalid.

If this operand is omitted, the current directory is assumed.

- `DCTRCERR=`*error-log-size* ~<unsigned integer> ((0-1073741824)) <<4096>> (unit: bytes)

  Specify the size of the error log files (`dcerr1.trc` and `dcerr2.trc`). These files are created in the directory specified with `DCTRCPATH` in the client environment definition, or in the directory where the CUP was executed.

  If you specify `0` or an invalid value, or if there is no information to be output, then error log information is not output.

  Although you can specify a maximum of 1 GB, when you have saved the error log trace in a file in an edited format, your machine may not be able to open such files, depending on the specifications of the machine. You must specify a valid value in accordance with the environment you are using.

- `DCTRCUAP=`*UAP-trace-file-size* ~<unsigned integer> ((4096-1073741824)) (unit: bytes)

  Specify the size of the UAP trace files (`dcuap1.trc` and `dcuap2.trc`). These files are created in the directory specified with `DCTRCPATH` in the client environment definition, or in the directory where the CUP was executed.

  If the specification is incorrect or omitted, or if there is no information to be output, UAP trace information is not output.

  Although you can specify a maximum of 1 GB, when you have saved the UAP trace in a file in an edited format, your machine may not be able to open such files, depending on the specifications of the machine. You must specify a valid value in accordance with the environment you are using.

- `DCTRCSOC=`*socket-trace-file-size* ~<unsigned integer> ((4096-1073741824)) <<do not output>> (unit: bytes)

  Specify the size of the socket trace files (`dcsoc1.trc` and `dcsoc2.trc`). These files are created in the directory specified with `DCTRCPATH` in the client environment definition, or in the directory where the CUP was executed.

  If the specification is incorrect or omitted, or if there is no information to be output, socket trace information is not output.

  Although you can specify a maximum of 1 GB, when you have saved the socket trace in a file in an edited format, your machine may not be able to open such files, depending on the specifications of the machine. You must specify a valid value in accordance with the environment you are using.

- `DCTRCSOCSIZE=`*socket-trace-data-size* ~<unsigned integer> ((64-4096)) <<256>> (unit: bytes)

Specify the data size of a socket trace to be output.

If the specification is incorrect or omitted, the default is validated.

■ DCTRCMDL=*module-trace-file-size* ~<unsigned integer> ((4096-1073741824)) <<do not output>> (unit: bytes)

Specify the size of the module trace files (dcmdl1.trc and dcmdl2.trc). These files are created in the directory specified with DCTRCPATH in the client environment definition, or in the directory where the CUP was executed.

If the specification is incorrect or omitted, or if there is no information to be output, module trace information is not output.

Although you can specify a maximum of 1 GB, when you have saved the module trace in a file in an edited format, your machine may not be able to open such files, depending on the specifications of the machine. You must specify a valid value in accordance with the environment you are using.

■ DCCLTPRFINFOSEND=Y|N ~<<N>>

Specify whether to send the identification information for a performance verification trace to TP1/Server.

Y: The identification information for a performance verification trace is sent to TP1/Server.

N: The identification information for a performance verification trace is not sent to TP1/Server.

When Y is specified for this operand and DCTRCUAP in the client environment definition is set to include performance verification trace information in the UAP trace, performance verification trace information is included in the UAP trace. If the identification information for the performance verification trace is sent to TP1/Server and the UAP trace is acquired, you can check the TP1/Client function execution time against the TP1/Server service execution time. In addition, you can also determine the progress of processing.

■ DCCLTRPCMAXMSGSIZE=*maximum-RPC-message-length*~<unsigned integer> ((1-8)) <<1>> (unit: megabytes)

Specify the maximum length of user data that can be sent and received by using an RPC or the facility for receiving one-way messages from the server. This definition is valid when the following functions and arguments are specified.

Function	Argument	
	Input parameter length	Output parameter length
dc_rpc_call_s function	in_len	out_len

Function	Argument	
	Input parameter length	Output parameter length
`dc_rpc_call_to_s` function	`in_len`	`out_len`
`dc_clt_accept_notification_s` function	--	`inf_len`
`dc_clt_cancel_notification_s` function	`inf_len`	--
`dc_clt_chained_accept_notification_s` function	--	`inf_len`

Legend:

-: Not applicable

If you specify 2 or a larger value in this operand, the maximum length of user data that can be sent and received is the value of this operand x 1024 x 1024 (bytes), rather than the value of `DCRPC_MAX_MESSAGE_SIZE`. If the specified value of an argument of one of the above functions is larger than the value of this operand x 1024 x 1024, the function returns a `DCRPCER_INVALID_ARGS` or `DCRPCER_MESSAGE_TOO_BIG` error.

If the length of received user data is larger than the value of the `inf_len` argument, the `dc_clt_accept_notification_s` function or `dc_clt_chained_accept_notification_s` function of the facility for receiving one-way messages returns a `DCCLTER_INF_TOO_BIG` error.

When specifying 2 or a larger value in this operand to send or receive user data larger than the value of `DCRPC_MAX_MESSAGE_SIZE`, note the following:

- You must also specify an appropriate value for the `rpc_max_message_size` operand in the system common definition for the TP1/Server to which you want to send a service request. Specify the same value for all the nodes specified for the `all_node` operand in the system common definition.
- If a request for a service is made to TP1/Server that does not support the `rpc_max_message_size` operand in the system common definition, operation is not guaranteed. If you execute either of the following RPCs, TP1/Server may go down:

  Specify `Y` for `DCSCDDIRECT` in the client environment definition and issue the `dc_rpc_call_s` function.

  Specify `N` for `DCCLTONLYTHISNODE` in the client environment definition and issue the `dc_rpc_call_to_s` function.
- If the response data length specified in the service function is larger than the

value of `rpc_max_message_size` in the system definition x 1024 x 1024, the `dc_rpc_call_s` function or `dc_rpc_call_to_s` function returns a `DCRPCER_INVALID_REPLY` error.

- If the TP1/Server inter-node load-balancing facility transfers a service request to another node, a `DCRPCER_NET_DOWN` error may be returned.
- When using the permanent connection establishment function or transaction control function, you must specify an appropriate value for the `rpc_max_message_size` operand in the system common definition. If you do not specify an appropriate value, the `dc_rpc_call_s` function returns a `DCRPCER_MESSAGE_TOO_BIG` or `DCRPCER_INVALID_ARGS` error.
- If the service is not running on TP1/Server that supports the `rpc_max_message_size` operand in the system common definition, the `dc_rpc_call_s` function or `dc_rpc_call_to_s` function returns a `DCRPCER_NO_SUCH_SERVICE_GROUP` or `DCRPCER_TRNCHK` error.

■ `DCCLTRECVBUFSIZE=`*TCP/IP-receive-buffer-size* ~<unsigned integer> ((8192-1048576)) (unit: bytes)

Specify the size of the TCP/IP receive buffer secured for each connection. Adjusting this value in relation to the buffer length of the remote system can improve the effectiveness of communication.

Note:

Upon receiving data, TCP returns a delivery confirmation packet (ACK).

However, if the amount of received data is less than the size of the receive buffer, TCP does not immediately return the ACK (the ACK is delayed).

In a communication environment in which the value of this operand is large and a small amount of data is sent and received, a delayed ACK might degrade performance. For details about delayed ACKs, see the TCP/IP documentation.

Make sure that the value of this operand is less than the size of the TCP/IP receive buffer available in the OS.

If this operand is omitted, the system default is used.

■ `DCCLTSENDBUFSIZE=`*TCP/IP-send-buffer-size* ~<unsigned integer> ((8192-1048576)) (unit: bytes)

Specify the size of the TCP/IP send buffer secured for each connection. Adjusting this value in relation to the buffer length of the remote system can improve the effectiveness of communication.

Note:

Make sure that the value of this operand is less than the size of the TCP/IP send buffer available in the OS.

If this operand is omitted, the system default is used.

- DCCLTTCPNODELAY=Y|N ~<<N>>

Specify whether to disable the Nagle algorithm.

Y: The Nagle algorithm is disabled.

N: The Nagle algorithm is not disabled.

Specifying Y for this operand might degrade sending efficiency during INET domain communication, increasing network load. Before specifying Y for this operand, carefully consider whether it is necessary to disable the Nagle algorithm in light of DCCLTSENDBUFSIZE and DCCLTRECVBUFSIZE operands in the client environment definition, and the network bandwidth.

For details about the Nagle algorithm, see the TCP/IP documentation.

7.2.6 TP1/Client/W-specific operands

- DCCLTBACKLOGCOUNT=*number-of-queues-for-storing-connection-establishment-requests* ~<unsigned integer> ((0-4096)) <<0>>

Specify the number of queues that store the connection establishment requests. If you specify 0 or omit the specification, the number of queues is as follows:

- For AIX 5L: 1024
- For Linux: 128
- For HI-UX/WE2: 20
- For HP-UX: 20
- For Solaris: 5

Note that the actual number of queues may be greater than the specified value. The upper and lower limits for the number of queues differ depending on the operating system. If the upper and lower limits for the number of queues are restricted by the operating system, the specified values may not become valid.

For details about queues that store connection establishment requests, see the documentation of the applicable operating system or the TCP/IP documentation.

This operand is invalid when the dc_clt_accept_notification_s function is executed.

If the operand is specified incorrectly, one of the following functions returns an error with DCCLTER_FATAL and the KFCA02401-E message is output to the error log.

- `dc_clt_cltin_s` function
- `dc_clt_open_notification_s` function
- `dc_clt_cancel_notification_s` function

7.2.7 Operands for TP1/Client/P only

■ `dcselint=`*reception-check-interval* ~<unsigned integer> ((0 - 65535)) <<100>> (unit: milliseconds)

Specify the interval for checking whether a response has been returned from the server.

When `0` is specified, the check result will be immediately acquired. However, other windows in the thread waiting for the response cannot operate until the server returns a response.

The value of the operand must be smaller than the value of `DCSYSWATCHTIM` and `DCWATCHTIM` in the client environment definition.

7.2.8 Notes on TP1/Client/W

- The environment variables are analyzed during `dc_clt_cltin_s` and `dc_rpc_open_s` function processing. Therefore, do not change the client environment definition after you call the `dc_clt_cltin_s` function. Similarly, do not change the client environment definition after calling the `dc_clt_open_notification_s` function.
- In TP1/Client/W, you can use a different client environment definition for each `dc_clt_cltin_s` function call. To do so, create a separate client environment definition file for each `dc_clt_cltin_s` function call, and specify the file name in the `defpath` argument of the function. This information also applies to the following functions:
    - `dc_clt_accept_notification_s`
    - `dc_clt_cancel_notification_s`
    - `dc_clt_open_notification_s`
- TP1/Client/W accesses the files specified in the `defpath` arguments so that the following functions can read the definitions:
    - `dc_clt_cltin_s`
    - `dc_clt_accept_notification_s`
    - `dc_clt_cancel_notification_s`
    - `dc_clt_open_notification_s`

  If the value of the `defpath` argument is NULL, the environment variables are loaded into TP1/Client/W. If a file is specified in the `defpath` argument, the client environment definitions specified as environment variables do not take effect.

7.2.9 Notes on TP1/Client/P

- The environment variables are analyzed during the processing of the `dc_clt_cltin_s` function and the `dc_rpc_open_s` function. Therefore, do not change the client environment definition after you call the `dc_clt_cltin_s` function. Similarly, do not change the client environment definition after calling the `dc_clt_open_notification_s` function.
- In TP1/Client/P, you can use a different client environment definition for each `dc_clt_cltin_s` function call. To do so, create a separate client environment definition file for each `dc_clt_cltin_s` function call, and specify the file name in the `defpath` argument of the function. This information also applies to the following functions:
    - `dc_clt_accept_notification_s`
    - `dc_clt_cancel_notification_s`
    - `dc_clt_open_notification_s`
- TP1/Client/P first accesses the `betran.ini` file in the Windows directory to read the definition. It then accesses the files specified in the `defpath` arguments so that the following functions can read the definition items. Definition items loaded later overwrite those loaded previously.
    - `dc_clt_cltin_s`
    - `dc_clt_accept_notification_s`
    - `dc_clt_cancel_notification_s`
    - `dc_clt_open_notification_s`

  If any definition items that are not specified in any of the files specified in the `defpath` arguments of the above functions are contained in the `betran.ini` file in the Windows directory, those definition items take effect.

Chapter 8. Operating Commands

This chapter explains how to code and use TP1/Client operating commands.

In this chapter, C functions (`dc_`*xxx*`_`*xxx*`_s`) when calling the DLLs are used in explanations. If you use functions of the normal object library (`dc_`*xxx*`_`*xxx*) or COBOL, replace the C function names with the corresponding functions or COBOL request statements.

This chapter contains the following sections:

8.1 Operating command syntax

Operating command syntax is shown below.

command-name∆*option*

command-name

Name of the command to be executed.

option

TP1/Client/W

A character string beginning with a minus sign (`-`) and taking no or one flag argument.

Option syntax is as follows.

`-`*option-flag*

`or`

`-`*option-flag*∆*flag-argument*

Legend:

option-flag: A single alphanumeric character

flag-argument: The argument for the option flag

TP1/Client/P

A character string beginning with a slash (`/`) and taking no or one flag argument.

Option syntax is as follows.

`/`*option-flag*

`or`

`/`*option-flag*∆*flag-argument*

Legend:

option-flag: A single alphanumeric character *flag-argument*: The argument for the option flag

8.2 Operating command descriptions

TP1/Client operating commands are described following.

8.2.1 cltdump (edit and output a trace)

(1) Form

When you use the `cltdump` command with TP1/Client/P, enter the command from the MS-DOS prompt.

(a) TP1/Client/W

`cltdump [-u|-s|-m|][-n][-f` *file-name*`]`

(b) TP1/Client/P

`cltdump32 [/u|/s|/m][/n][/f` *file-name*`]`

(2) Purpose

Edits the trace of TP1/Client and outputs the edited data to the standard output. To save the edited data in a file, redirect the standard output to a file. This command can edit the following traces:

- UAP trace (`dcuap1.trc` and `dcuap2.trc`)
- Socket trace (`dcsoc1.trc` and `dcsoc2.trc`)
- Module trace (`dcmdl1.trc` and `dcmdl2.trc`)

Each trace is stored in two files. The `cltdump` command first outputs the data in the older file.

Although the contents of the socket trace and module trace are not disclosed, maintenance personnel may use them for troubleshooting.

Before using this command, make sure that the version of the command is the same as the version of TP1/Client that outputs the trace you want to edit and output. If you use the command of a different version, the editing results may be incorrect.

(3) Option

- `-u` or `/u`

  Use this option to edit and output the UAP trace (`dcuap1.trc` and `dcuap2.trc`).

- `-s` or `/s`

Use this option to edit and output the socket trace (`dcsoc1.trc` and `dcsoc2.trc`). The contents of the socket trace are not disclosed.

- `-m` or `/m`

  Use this option to edit and output the module trace (`dcmdl1.trc` and `dcmdl2.trc`). The contents of the module trace are not disclosed.

- `-n` or `/n`

  You can add this option when editing and outputting the UAP trace. Use this option when you want to output function names for user-issued functions. If you do not use this option, the command outputs function codes for user-issued functions.

  This option takes effect only when you edit and output the UAP trace.

- `-f` *file-name* or `/f` *file-name*

  Use this option to output a specific trace file to the standard output. Specify the full path name or file name. The file is assumed to be in the current directory if the file name only is specified.

  When you specify this option, only the specified file is edited and output, and the two files (for a UAP trace, `dcuap1.trc` and `dcuap2.trc`) are not merged.

  When you omit this option, the two files (for a UAP trace, `dcuap1.trc` and `dcuap2.trc`) that are output to the CUP executing directory or the directory specified using `DCTRCPATH` in the client environment definition are merged, edited, and output.

(4) Output example

The contents of the socket trace and module trace are not disclosed. The following shows an output example of the UAP trace.

```
DATE = aaaaaaaaaa TIME = bbbbbbbbbbb       PID = ccccc     SIZE = ddd
CODE1 = eeeeeeee    CODE2 = f           RETURN = ggggg
ADDRESS   +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +a +b +c +d +e +f  0123456789abcdef  ┐
00000000  73 61 69 73 70 70 00 00 00 00 00 00 00 00 00 00  saispp..........  │
00000010  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................  │ 1
00000020  73 61 69 73 76 72 30 31 00 00 00 00 00 00 00 00  saisvr01........  │
   :                             :                                 :          ┘
└───────┘ └──────────────────────────────────────────────┘ └──────────────┘
    2                            3                                 4
```

aaaaaaaaaa

UAP trace date

Format: *yyyy*/*mm*/*dd* (year/month/day)

bbbbbbbbb

UAP trace time

Format: *hh*:*mm*:*ss*:*sss*(hour:minute:second:millisecond)

ccccc

Process ID of the process that collected the UAP trace.

When TP1/Client/W is used, the process ID is displayed in the *process-ID*:*client-ID* format.

When TP1/Client/P is used, the process ID is displayed in the *process-ID*:*thread-ID* format.

ddd

Data size

eeeeeeee

Type code 1 of the called function

180000 (hexadecimal): Request from TP1/Client

f

Type code 2 of the called function

`1`: `dc_rpc_open_s` function

2: `dc_rpc_close_s` function

3: `dc_rpc_call_s` function

4: `dc_clt_cltin_s` function

5: `dc_clt_cltout_s` function

6: `dc_clt_send_s` function

7: `dc_clt_receive_s` function

8: `dc_trn_begin_s` function

9: `dc_trn_chained_commit_s` function

1a: `dc_clt_set_raphost_s` function

1b: `dc_clt_get_raphost_s` function

1c: `dc_clt_assem_send_s` function

1d: `dc_clt_assem_receive_s` function

a: `dc_trn_chained_rollback_s` function

b: `dc_trn_unchained_commit_s` function

c: `dc_trn_unchained_rollback_s` function

d: `dc_trn_info_s` function

e: `dc_clt_get_trnid_s` function

f: `dc_rpc_get_watch_time_s` function

10: `dc_rpc_set_watch_time_s` function

13: `dc_clt_connect_s` function

14: `dc_clt_disconnect_s` function

17: `dc_clt_receive2_s` function

18: `dc_clt_set_connect_inf_s` function

19: `dc_rpc_call_to_s` function

100: `dc_clt_accept_notification_s` function

101: `dc_clt_cancel_notification_s` function

102: `dc_clt_open_notification_s` function

103: `dc_clt_close_notification_s` function

104: `dc_clt_chained_accept_notification_s` function

200: `tpalloc` function

`201`: `tpfree` function

`202`: `tpconnect` function

`203`: `tpdiscon` function

`204`: `tpsend` function

`205`: `tprecv` function

ggggg

Return code of the called function (decimal)

See Section *4. TP1/Client Functions (C Language)* or *Section 5.1 COBOL-UAP creation program features* for code values.

1. Call information for the OpenTP1 function#
2. Address of call information for the OpenTP1 function
3. Hexadecimal display of call information for the OpenTP1 function
4. ASCII character display of call information for the OpenTP1 function

#

For the format of call information of each function, see (5).

(5) Format of call information

Table 8-1: Call information for the dc_rpc_open_s function (function code: 1)

Type	Length (bytes)		Area name	Position (hex.)	Description
Function entrance information	8	4	`cltid`	0	The value of the `cltid` argument for the `dc_clt_open_s` function. When the `dc_clt_open` function is used, the area is cleared to 0.
		4	`flags`	4	The value of the `flags` argument for the `dc_clt_open_s` function

Note:

There is no function exit information.

Table 8-2: Call information for the dc_rpc_close_s function (function code: 2)

Type	Length (bytes)		Area name	Position (hex.)	Description
Function entrance information	8	4	`cltid`	0	The value of the `cltid` argument for the `dc_rpc_close_s` function. When the `dc_rpc_close` function is used, the area is cleared to 0.
		4	`flags`	4	The value of the `flags` argument for the `dc_rpc_close_s` function

Note:

There is no function exit information.

Table 8-3: Call information for the dc_rpc_call_s function (function code: 3)

Type	Length (bytes)		Area name	Position (hex.)	Description
Function entrance information	140	4	`cltid`	0	The value of the `cltid` argument for the `dc_rpc_call_s` function. When the `dc_rpc_call` function is used, the area is cleared to 0.
		32	`group`	4	The value of the `group` argument for the `dc_rpc_call_s` function
		32	`service`	24	The value of the `service` argument for the `dc_rpc_call_s` function
		60	`in`	44	The value of the `in` argument for the `dc_rpc_call_s` function
		4	`in_len`	80	The value of the `in_len` argument for the `dc_rpc_call_s` function
		4	`out_len`	84	The value of the `out_len` argument for the `dc_rpc_call_s` function
		4	`flags`	88	The value of the `flags` argument for the `dc_rpc_call_s` function
Function exit information	64	60	`out`	0	The value returned to the `out` argument for the `dc_rpc_call_s` function
		4	`out_len`	3C	The value returned to the `out_len` argument for the `dc_rpc_call_s` function

Table 8-4: Call information for the dc_clt_cltin_s function (function code: 4)

Type	Length (bytes)		Area name	Position (hex.)	Description
Function entrance information	168	4	`hWnd`	0	The value of the `hWnd` argument for the `dc_clt_cltin_s` function. When the `dc_clt_cltin` function is used, the area is cleared to 0.
		64	`defpath`	4	The value of the `defpath` argument for the `dc_clt_cltin_s` function. When the `dc_clt_cltin` function is used, the area is cleared to 0.
		64	`target_host`	44	The value of the `target_host` argument for the `dc_clt_cltin_s` function
		32	`logname`	84	The value of the `logname` argument for the `dc_clt_cltin_s` function
		4	`flags`	A4	The value of the `flags` argument for the `dc_clt_cltin_s` function
Function exit information	68	4	`cltid`	0	The value returned to the `cltid` argument for the `dc_clt_cltin_s` function. When the `dc_clt_cltin` function is used, the area is cleared to 0.
		64	`set_host`	4	The value returned to the `set_host` argument for the `dc_clt_cltin_s` function

Table 8-5: Call information for the dc_clt_cltout_s function (function code: 5)

Type	Length (bytes)		Area name	Position (hex.)	Description
Function entrance information	8	4	`cltid`	0	The value of the `cltid` argument for the `dc_clt_cltout_s` function. When the `dc_clt_cltout` function is used, the area is cleared to 0.
		4	`flags`	4	The value of the `flags` argument for the `dc_clt_cltout_s` function

Note:

There is no function exit information.

Table 8-6: Call information for the dc_clt_send_s function (function code: 6)

Type	Length (bytes)		Area name	Position (hex.)	Description
Function entrance information	144	4	`cltid`	0	The value of the `cltid` argument for the `dc_clt_send_s` function. When the `dc_clt_send` function is used, the area is cleared to 0.
		64	`buff`	4	The value of the `buff` argument for the `dc_clt_send_s` function
		4	`sendleng`	44	The value of the `sendleng` argument for the `dc_clt_send_s` function
		64	`hostname`	48	The value of the `hostname` argument for the `dc_clt_send_s` function
		2	`portnum`	88	The value of the `portnum` argument for the `dc_clt_send_s` function
		2	-	8A	Reserved area
		4	`flags`	8C	The value of the `flags` argument for the `dc_clt_send_s` function.

Legend:

-: Not applicable.

Note:

There is no function exit information.

Table 8-7: Call information for the dc_clt_receive_s function (function code: 7)

Type	Length (bytes)		Area name	Position (hex.)	Description
Function entrance information	16	4	`cltid`	0	The value of the `cltid` argument for the `dc_clt_receive_s` function. When the `dc_clt_receive` function is used, the area is cleared to 0.
		4	`recvleng`	4	The value of the `recvleng` argument for the `dc_clt_receive_s` function
		4	`timeout`	8	The value of the `timeout` argument for the `dc_clt_receive_s` function
		4	`flags`	C	The value of the `flags` argument for the `dc_clt_receive_s` function

Type	Length (bytes)		Area name	Position (hex.)	Description
Function exit information	64	64	`buff`	0	The value returned to the `buff` argument for the `dc_clt_receive_s` function.

Table 8-8: Call information for the dc_trn_begin_s function (function code: 8)

Type	Length (bytes)		Area name	Position (hex.)	Description
Function entrance information	4	4	`cltid`	0	The value of the `cltid` argument for the `dc_trn_begin_s` function When the `dc_trn_begin` function is used, the area is cleared to 0.
Function exit information	32	16	`trngid`	0	The transaction global identifier for the global transaction that occurred
		16	`trnbid`	10	The transaction branch identifier for the global transaction that occurred

Table 8-9: Call information for the dc_trn_chained_commit_s function (function code: 9)

Type	Length (bytes)		Area name	Position (hex.)	Description
Function entrance information	36	4	`cltid`	0	The value of the `cltid` argument for the `dc_trn_chained_commit_s` function When the `dc_trn_chained_commit` function is used, the area is cleared to 0.
		16	`trngid`	4	The transaction global identifier for the current global transaction
		16	`trnbid`	14	The transaction branch identifier for the current global transaction
Function exit information	32	16	`trngid`	0	The transaction global identifier for a new global transaction that occurred
		16	`trnbid`	10	The transaction branch identifier for a new global transaction that occurred

Table 8-10: Call information for the dc_clt_set_raphost_s function (function code: 1a)

Type	Length (bytes)		Area name	Position (hex.)	Description
Function entrance information	136	4	`cltid`	0	The value of the `cltid` argument for the `dc_clt_set_raphost_s` function When the `dc_clt_set_raphost` function is used, the area is cleared to 0.
		128	`raphost`	4	The value of the `raphost` argument for the `dc_clt_set_raphost_s` function
		4	`flags`	84	The value of the `flags` argument for the `dc_clt_set_raphost_s` function

Note:

There is no function exit information.

Table 8-11: Call information for the dc_clt_get_raphost_s function (function code: 1b)

Type	Length (bytes)		Area name	Position (hex.)	Description
Function entrance information	8	4	`cltid`	0	The value of the `cltid` argument for the `dc_clt_get_raphost_s` function When the `dc_clt_get_raphost` function is used, the area is cleared to 0.
		4	`flags`	4	The value of the `flags` argument for the `dc_clt_get_raphost_s` function
Function exit information	128	128	`raphost`	0	The value returned to the `raphost` argument for the `dc_clt_get_raphost_s` function.

Table 8-12: Call information for the dc_clt_assem_send_s function (function code: 1c)

Type	Length (bytes)		Area name	Position (hex.)	Description
Function entrance information	148	4	`cltid`	0	The value of the `cltid` argument for the `dc_clt_assem_send_s` function When the `dc_clt_assem_send` function is used, the area is used by TP1/Client.
		64	`buff`	4	The value of the `buff` argument for the `dc_clt_assem_send_s` function

Type	Length (bytes)		Area name	Position (hex.)	Description
		4	`sendleng`	44	The value of the `sendleng` argument for the `dc_clt_assem_send_s` function
		64	`hostname`	48	The value of the `hostname` argument for the `dc_clt_assem_send_s` function
		2	`portnum`	88	The value of the `portnum` argument for the `dc_clt_assem_send_s` function
		2	--	8A	Unused area
		4	`timeout`	8C	The value of the `timeout` argument for the `dc_clt_assem_send_s` function
		4	`flags`	90	The value of the `flags` argument for the `dc_clt_assem_send_s` function

Legend:

--: Not applicable.

Note:

There is no function exit information.

Table 8-13: Call information for the dc_clt_assem_receive_s function (function code: 1d)

Type	Length (bytes)		Area name	Position (hex.)	Description
Function entrance information	16	4	`cltid`	0	The value of the `cltid` argument for the `dc_clt_assem_receive_s` function When the `dc_clt_assem_receive` function is used, the area is used by TP1/Client.
		4	`recvleng`	4	The value of the `recvleng` argument for the `dc_clt_assem_receive_s` function
		4	`timeout`	8	The value of the `timeout` argument for the `dc_clt_assem_receive_s` function
		4	`flags`	C	The value of the `flags` argument for the `dc_clt_assem_receive_s` function
Function exit information	68	64	`buff`	0	The value returned to the `buff` argument for the `dc_clt_assem_receive_s` function
		4	`recvleng`	40	The value returned to the `recvleng` argument for the `dc_clt_assem_receive_s` function

Table 8-14: Call information for the dc_trn_chained_rollback_s function (function code: a)

Type	Length (bytes)		Area name	Position (hex.)	Description
Function entrance information	36	4	`cltid`	0	The value of the `cltid` argument for the `dc_trn_chained_rollback_s` function. When the `dc_trn_chained_rollback` function is used, the area is cleared to 0.
		16	`trngid`	4	The transaction global identifier for the current global transaction
		16	`trnbid`	14	The transaction branch identifier for the current global transaction
Function exit information	32	16	`trngid`	0	The transaction global identifier for a new global transaction that occurred
		16	`trnbid`	10	The transaction branch identifier for a new global transaction that occurred

Table 8-15: Call information for the dc_trn_unchained_commit_s function (function code: b)

Type	Length (bytes)		Area name	Position (hex.)	Description
Function entrance information	36	4	`cltid`	0	The value of the `cltid` argument for the `dc_trn_unchained_commit_s` function. When the `dc_trn_unchained_commit` function is used, the area is cleared to 0.
		16	`trngid`	4	The transaction global identifier for the current global transaction
		16	`trnbid`	14	The transaction branch identifier for the current global transaction

Note:

There is no function exit information.

Table 8-16: Call information for the dc_trn_unchained_rollback_s function (function code: c)

Type	Length (bytes)		Area name	Position (hex.)	Description
Function entrance information	36	4	`cltid`	0	The value of the `cltid` argument for the `dc_trn_unchained_rollback_s` function. When the `dc_trn_unchained_rollback` function is used, the area is cleared to 0.
		16	`trngid`	4	The transaction global identifier for the current global transaction
		16	`trnbid`	14	The transaction branch identifier for the current global transaction

Note:

There is no function exit information.

Table 8-17: Call information for the dc_trn_info_s function (function code: d)

Type	Length (bytes)		Area name	Position (hex.)	Description
Function entrance information	8	4	`cltid`	0	The value of the `cltid` argument for the `dc_trn_info_s` function When the `dc_trn_info` function is used, the area is cleared to 0.
		4	`flags`	4	The value of the `flags` argument for the `dc_trn_info_s` function (address)

Note:

There is no function exit information.

Table 8-18: Call information for the dc_clt_get_trnid_s function (function code: e)

Type	Length (bytes)		Area name	Position (hex.)	Description
Function entrance information	4	4	`cltid`	0	The value of the `cltid` argument for the `dc_clt_get_trnid_s` function When the `dc_clt_get_trnid` function is used, the area is cleared to 0.

Type	Length (bytes)		Area name	Position (hex.)	Description
Function exit information	32	16	`trngid`	0	The value returned to the `trngid` argument for the `dc_clt_get_trnid_s` function.
		16	`trnbid`	10	The value returned to the `trnbid` argument for the `dc_clt_get_trnid_s` function.

Table 8-19: Call information for the dc_rpc_get_watch_time_s function (function code: f)

Type	Length (bytes)		Area name	Position (hex.)	Description
Function entrance information	4	4	`cltid`	0	The value of the `cltid` argument for the `dc_rpc_get_watch_time_s` function When the `dc_rpc_get_watch_time` function is used, the area is cleared to 0.

Note:

There is no function exit information.

Table 8-20: Call information for the dc_rpc_set_watch_time_s function (function code: 10)

Type	Length (bytes)		Area name	Position (hex.)	Description
Function entrance information	8	4	`cltid`	0	The value of the `cltid` argument for the `dc_rpc_set_watch_time_s` function When the `dc_rpc_set_watch_time` function is used, the area is cleared to 0.
		4	`var`	4	The value of the `var` argument for the `dc_rpc_set_watch_time_s` function

Note:

There is no function exit information.

Table 8-21: Call information for the dc_clt_connect_s function (function code: 13)

Type	Length (bytes)		Area name	Position (hex.)	Description
Function entrance information	8	4	`cltid`	0	The value of the `cltid` argument for the `dc_clt_connect_s` function When the `dc_clt_connect` function is used, the area is cleared to 0.
		4	`flags`	4	The value of the `flags` argument for the `dc_clt_connect_s` function

Note:

There is no function exit information.

Table 8-22: Call information for the dc_clt_disconnect_s function (function code: 14)

Type	Length (bytes)		Area name	Position (hex.)	Description
Function entrance information	8	4	`cltid`	0	The value of the `cltid` argument for the `dc_clt_disconnect_s` function When the `dc_clt_disconnect` function is used, the area is cleared to 0.
		4	`flags`	4	The value of the `flags` argument for the `dc_clt_disconnect_s` function

Note:

There is no function exit information.

Table 8-23: Call information for the dc_clt_receive2_s function (function code: 17)

Type	Length (bytes)		Area name	Position (hex.)	Description
Function entrance information	16	4	`cltid`	0	The value of the `cltid` argument for the `dc_clt_receive2_s` function When the `dc_clt_receive2` function is used, the area is cleared to 0.
		4	`recvleng`	4	The value of the `recvleng` argument for the `dc_clt_receive2_s` function
		4	`timeout`	8	The value of the `timeout` argument for the `dc_clt_receive2_s` function
		4	`flags`	C	The value of the `flags` argument for the `dc_clt_receive2_s` function
Function exit information	68	64	`buff`	0	The value returned to the `buff` argument for the `dc_clt_receive2_s` function.
		4	`recvleng`	40	The value returned to the `recvleng` argument for the `dc_clt_receive2_s` function.

Table 8-24: Call information for the dc_clt_set_connect_inf_s function (function code: 18)

Type	Length (bytes)		Area name	Position (hex.)	Description
Function entrance information	76	4	`cltid`	0	The value of the `cltid` argument for the `dc_clt_set_connect_inf_s` function
		2	`inf_len`	4	The value of the `inf_len` argument for the `dc_clt_set_connect_inf_s` function
		2	-	6	Reserved area
		64	`inf`	8	The value of the `inf` argument for the `dc_clt_set_connect_inf_s` function
		4	`flags`	48	The value of the `flags` argument for the `dc_clt_set_connect_inf_s` function

Legend:

-: Not applicable.

Note:

There is no function exit information.

Table 8-25: Call information for the dc_rpc_call_to_s function (function code: 19)

Type	Length (bytes)		Area name	Position (hex.)	Description
Function entrance information	236	4	`cltid`	0	The value of the `cltid` argument for the `dc_rpc_call_to_s` function When the `dc_rpc_call`_to function is used, the area is cleared to 0.
		8	`tbl_nid`	4	Reserved area (cleared to 0)
		64	`tbl_hostnm`	C	Value of the `hostnm` member of the `DCRPC_BINDING_TBL` structure
		2	`tbl_portno`	4C	Value of the `portno` member of the `DCRPC_BINDING_TBL` structure
		2	`tbl_filler1`	4E	Reserved area
		4	`tbl_flags`	50	Reserved area (0x00000002)
		16	`tbl_filler2`	54	Reserved area
		32	`group`	64	The value of the `group` argument for the `dc_rpc_call_to_s` function
		32	`service`	84	The value of the `service` argument for the `dc_rpc_call_to_s` function
		60	`in`	A4	The value of the `in` argument for the `dc_rpc_call_to_s` function
		4	`in_len`	E0	The value of the `in_len` argument for the `dc_rpc_call_to_s` function
		4	`out_len`	E4	The value of the `out_len` argument for the `dc_rpc_call_to_s` function
		4	`flags`	E8	The value of the `flags` argument for the `dc_rpc_call_to_s` function
Function exit information	64	60	`out`	0	The value returned to the `out` argument for the `dc_rpc_call_s` function.
		4	`out_len`	3C	The value returned to the `out_len` argument for the `dc_rpc_call_s` function.

Table 8-26: Call information for the dc_clt_accept_notification_s function (function code: 100)

Type	Length (bytes)		Area name	Position (hex.)	Description
Function entrance information	84	4	`hWnd`	0	The value of the `hWnd` argument for the `dc_clt_accept_notification_s` function When the `dc_clt_accept_notification` function is used, the area is cleared to 0.
		64	`defpath`	4	The value of the `defpath` argument for the `dc_clt_accept_notification_s` function When the `dc_clt_accept_notification` function is used, the area is cleared to 0.
		4	`inf_len`	44	The value of the `inf_len` argument for the `dc_clt_accept_notification_s` function
		2	`port`	48	The value of the `port` argument for the `dc_clt_accept_notification_s` function
		2	-	4A	Reserved area
		4	`timeout`	4C	The value of the `timeout` argument for the `dc_clt_accept_notification_s` function
		4	`flags`	50	The value of the `flags` argument for the `dc_clt_accept_notification_s` function
Function exit information	140	64	`inf`	0	The value returned to the `inf` argument for the `dc_clt_accept_notification_s` function.
		4	`inf_len`	40	The value returned to the `inf_len` argument for the `dc_clt_accept_notification_s` function.
		64	`hostname`	44	The value returned to the `hostname` argument for the `dc_clt_accept_notification_s` function.
		8	`nodeid`	84	The value returned to the `nodeid` argument for the `dc_clt_accept_notification_s` function.

Legend:

-: Not applicable.

Table 8-27: Call information for the dc_clt_cancel_notification_s function (function code: 101)

Type	Length (bytes)		Area name	Position (hex.)	Description
Function entrance information	208	4	hWnd	0	The value of the hWnd argument for the dc_clt_cancel_notification_s function When the dc_clt_cancel_notification function is used, the area is cleared to 0.
		64	defpath	4	The value of the defpath argument for the dc_clt_cancel_notification_s function When the dc_clt_cancel_notification function is used, the area is cleared to 0.
		64	inf	44	The value of the inf argument for the dc_clt_cancel_notification_s function
		4	inf_len	84	The value of the inf_len argument for the dc_clt_cancel_notification_s function
		2	port	88	The value of the port argument for the dc_clt_cancel_notification_s function
		2	-	8A	Reserved area
		64	hostname	8C	The value of the hostname argument for the dc_clt_cancel_notification_s function
		4	flags	CC	The value of the flags argument for the dc_clt_cancel_notification_s function

Legend:

-: Not applicable.

Note:

There is no function exit information.

Table 8-28: Call information for the dc_clt_open_notification_s function (function code: 102)

Type	Length (bytes)		Area name	Position (hex.)	Description
Function entrance information	76	4	`hWnd`	0	The value of the `hWnd` argument for the `dc_clt_open_notification_s` function When the `dc_clt_open_notification` function is used, the area is cleared to 0.
		64	`defpath`	4	The value of the `defpath` argument for the `dc_clt_open_notification_s` function When the `dc_clt_open_notification` function is used, the area is cleared to 0.
		2	`port`	44	The value of the `port` argument for the `dc_clt_open_notification_s` function
		2	-	46	Reserved area
		4	`flags`	48	The value of the `flags` argument for the `dc_clt_open_notification_s` function
Function exit information	4	4	`ntfid`	0	The value returned to the `ntfid` argument for the `dc_clt_open_notification_s` function.

Legend:

-: Not applicable.

Table 8-29: Call information for the dc_clt_close_notification_s function (function code: 103)

Type	Length (bytes)		Area name	Position (hex.)	Description
Function entrance information	8	4	`ntfid`	0	The value of the `ntfid` argument for the `dc_clt_close_notification_s` function When the `dc_clt_close_notification` function is used, the area is cleared to 0.
		4	`flags`	4	The value of the `flags` argument for the `dc_clt_close_notification_s` function

Note:

There is no function exit information.

Table 8-30: Call information for the dc_clt_chained_accept_notification_s function (function code: 104)

Type	Length (bytes)		Area name	Position (hex.)	Description
Function entrance information	16	4	`ntfid`	0	The value of the `ntfid` argument for the `dc_clt_chained_accept_notification_s` function When the `dc_clt_chained_accept_notification` function is used, the area is cleared to 0.
		4	`inf_len`	4	The value of the `inf_len` argument for the `dc_clt_chained_accept_notification_s` function
		4	`timeout`	8	The value of the `timeout` argument for the `dc_clt_chained_accept_notification_s` function
		4	`flags`	C	The value of the `flags` argument for the `dc_clt_chained_accept_notification_s` function
Function exit information	140	64	`inf`	0	The value returned to the `inf` argument for the `dc_clt_accept_notification_s` function.
		4	`inf_len`	40	The value returned to the `inf_len` argument for the `dc_clt_accept_notification_s` function.
		64	`hostname`	44	The value returned to the `hostname` argument for the `dc_clt_accept_notification_s` function.
		8	`nodeid`	84	The value returned to the `nodeid` argument for the `dc_clt_accept_notification_s` function.

Table 8-31: Call information for the tpalloc function (function code: 200)

Type	Length (bytes)		Area name	Position (hex.)	Description
Function entrance information	20	8	`type`	0	The value of the `type` argument for the `tpalloc` function
		8	`subtype`	8	The value of the `subtype` argument for the `tpalloc` function
		4	`size`	10	The value of the `size` argument for the `tpalloc` function

Type	Length (bytes)		Area name	Position (hex.)	Description
Function exit information	8	4	`return`	0	The value returned by the `tpalloc` function (address)
		4	`tperrno`	4	Value set in `tperrno`

Table 8-32: Call information for the tpfree function (function code: 201)

Type	Length (bytes)		Area name	Position (hex.)	Description
Function entrance information	4	4	`ptr`	0	The value of the `ptr` argument for the `tpfree` function (address)

Note:

There is no function exit information.

Table 8-33: Call information for the tpconnect function (function code: 202)

Type	Length (bytes)		Area name	Position (hex.)	Description
Function entrance information	100	32	`svc`	0	The value of the `svc` argument for the `tpconnect` function
		60	`data`	20	The value of the `data` argument for the `tpconnect` function
		4	`len`	5C	The value of the `len` argument for the `tpconnect` function
		4	`flags`	60	The value of the `flags` argument for the `tpconnect` function
Function exit information	4	4	`tperrno`	0	Value set in `tperrno`

Table 8-34: Call information for the tpdiscon function (function code: 203)

Type	Length (bytes)		Area name	Position (hex.)	Description
Function entrance information	4	4	`cd`	0	The value of the `cd` argument for the `tpdiscon` function
Function exit information	4	4	`tperrno`	0	Value set in `tperrno`

Table 8-35: Call information for the tpsend function (function code: 204)

Type	Length (bytes)		Area name	Position (hex.)	Description
Function entrance information	72	4	`cd`	0	The value of the `cd` argument for the `tpsend` function
		60	`data`	4	The value of the `data` argument for the `tpsend` function
		4	`len`	40	The value of the `len` argument for the `tpsend` function
		4	`flags`	44	The value of the `flags` argument for the `tpsend` function
Function exit information	12	4	`revent`	0	The value returned to the `revent` argument for the `tpsend` function.
		4	`tpurcode`	4	Value set in `tpurcode`
		4	`tperrno`	8	Value set in `tperrno`

Table 8-36: Call information for the tprecv function (function code: 205)

Type	Length (bytes)		Area name	Position (hex.)	Description
Function entrance information	12	4	`cd`	0	The value of the `cd` argument for the `tprecv` function
		4	`len`	4	The value of the `len` argument for the `tprecv` function
		4	`flags`	8	The value of the `flags` argument for the `tprecv` function
Function exit information	76	60	`data`	0	The value returned to the `data` argument for the `tprecv` function.
		4	`len`	3C	The value returned to the `len` argument for the `tprecv` function.
		4	`revent`	40	The value returned to the `revent` argument for the `tprecv` function.
		4	`tpurcode`	44	Value set in `tpurcode`
		4	`tperrno`	48	Value set in `tperrno`

Chapter

9. Error Recovery

This chapter outlines measures to be taken in the event of an error.

This chapter contains the following sections:

9.1 Communication errors

This section provides notes on communication errors in TP1/Client and how they are handled.

- Output of an error message when a communication error occurs

  TP1/Client uses TPC, a connection-type communication protocol, and implements it with a socket interface. TP1/Client therefore detects communication errors as system call errors at the sockets interface. If a system call error occurs, TP1/Client outputs an error message to identify the error cause.

  Error messages are output to error log files. For details on error log files, see *2.11.1 Error logging*.

- Communication error related to ports

  TP1/Client establishes and releases a connection each time communication occurs. If multiple CUPs repeatedly issue the `dc_clt_cltin_s` function and `dc_clt_cltout_s` function or issue processing in succession that repeatedly performs the `dc_rpc_call_s` function, an error may occur because the ports of the operating system are all temporarily being used. If such an error occurs, increase the number of ports to tune the setting to a value acceptable for operation. Alternatively, wait until ports are released and ready for operation, and then retry the communication.

- Communication error that occurs when the `dc_rpc_call_s` function is used

  If the `dc_rpc_call_s` function is issued while the communication destination schedule service is starting or terminating, the system switches to another host to perform the processing. If one of the hosts specified in the `target_host` argument of the `dc_clt_cltin_s` function or `DCHOST` in the client environment definition has not started, the `dc_rpc_call_s` function may require some time until it returns.

  If the `dc_rpc_call_s` function is issued during establishment of a permanent connection or within a transaction, the host is not switched when an error response is received.

9.2 Client errors

PC processing can be stopped by simple causes such as power failure or misoperation. Data cannot be guaranteed if such problems occur during processing of an RPC execution request, even if a response has been received from the SPP.

9.3 Errors in a remote operation request to XDM/DCCM3

(1) DCRPCER_NET_DOWN during a remote operation request to XDM/DCCM3

When you issue a remote operation request to XDM/DCCM3, not only a network failure but also the following conditions cause `DCRPCER_NET_DOWN`.

- Incorrect message format
- Incorrect transaction name
- Invalid combination of the transaction type and the RPC call type
- Invalid combination of the transaction type and the virtual terminal type
- Inhibited transaction input
- Transaction shutdown
- Security error
- Input queue error

(2) Duplicate input error on XDM/DCCM3

When the same personal computer or workstation uses multiple CUPs on TP1/Client, you may issue a synchronous-response type RPC to an XDM/DCCM3 terminal that is dedicated to the remote host computer and concurrently works as a send/receive terminal. In this situation, specifying an XDM/DCCM3 transaction to be inquiry-response type causes a duplicate input error. As a solution, specify the transaction type to be no-inquiry-response type.

Chapter

10. Messages

This chapter describes the messages output by TP1/Client.

In this chapter, C functions (`dc_`*xxx*`_`*xxx*`_s`) when calling the DLLs are used in explanations. If you use functions of the normal object library (`dc_`*xxx*`_`*xxx*) or COBOL, replace the C function names with the corresponding functions or COBOL request statements.

This chapter contains the following sections:

10.1 Format of output messages

Messages are output in the following format:

KFCA00000-X

```
YY...YY
```

`KFCA00000-X`

Message ID (11 half-size alphanumerics)

YY...YY

Message text

10.2 Format of message descriptions

This section shows the format of message descriptions and explains the elements in a message ID.

(1) Description format

This subsection shows the format of message descriptions in this manual.

KFCA$n_1n_2n_3n_4n_5$-X (Y)

`Message text`

An explanation of the message follows the message text.

S: The main processing performed by the system after message output.

O: Action to be taken by the operator.

Countermeasure:

Action to be taken by the TP1/Client system administrator.

Note

"Contact maintenance personnel" in the message text, operator action, or recovery section indicates that Hitachi staff or the Hitachi Sales Division should be contacted.

(2) Message IDs

The meanings of the message IDs are as follows:

`KFCA`

OpenTP1 message

n1n2n3n4n5

Message sequence number

`X`

Message type (Table 10-1)

Table 10-1: Message types

Type	Meaning
`E`	The error disables the TP1/Client library and operating commands.
	The operation cannot be performed due to an invalid definition or incorrectly specified operand in the operating command.

Type	Meaning
`W`	Even though there is an invalid definition or incorrectly specified operand in the command, the operation continues assuming a correct value.
`I`	Information messages which do not fit the categories described in E, W above and simply report an operation.

`Y`

Message output destination. If there are two or more destinations, the possible destination codes are combined with a plus (+) sign. (Table 10-2)

Table 10-2: Message output destinations

Type	Output Destination
`E`	Standard error output
`S`	Standard output
`R`	Error log file

10.3 List of messages

Messages output by TP1/Client are described in message number order.

KFCA02401-E (R)

`environment definition error, variable=`*aa...aa*`, reason=`*bb...bb*

There is a specification error in the client environment definition.

aa...aa

Name of the incorrect operand in the client environment definition

bb...bb

Reason

`NO VALUE`

No value is specified.

`OUT OF RANGE`

The specified value is out of range.

`INVALID CHAR`

The specified value includes an invalid character.

`NET ENVIRONMENT`

The specified value is inconsistent with the network environment.

`UNMATCH LENG`

The number of digits in the specified value is invalid.

S: Stops processing, or continues processing by using defaults.

O: Contact the TP1/Client system administrator.

Countermeasure:

Correct the definition error.

KFCA02402-E (R+E)

`memory shortage, size=`*aa...aa*`, inf=`*bb...bb*

Memory with the number of bytes shown at "size" cannot be allocated.

aa...aa

Number of bytes that should have been allocated

bb...bb

Maintenance information

S: Stops processing.

O: Contact the TP1/Client system administrator.

Countermeasure:

Re-estimate the memory requirement. When memory is sufficient, collect the error log and traces, and contact maintenance personnel.

KFCA02403-E (R)

aa...aa `version error, TP1/Client(`*bb...bb*`):object(`*cc...cc*`)`

Processing is impossible because the version of *aa...aa* OpenTP1 program is inconsistent with that of TP1/Client library linked to CUP.

aa...aa

Program whose version is inconsistent

NAM

NAM of TP1/Server Base

bb...bb

Version of TP1/Client library linked to CUP

cc...cc

Version of OpenTP1 program

S: Stops processing.

O: Contact the TP1/Client system administrator.

Countermeasure:

- For *bb...bb* < *cc...cc*

  Relink CUP.

- For *bb...bb* > *cc...cc*

  Check if TP1/Client is installed correctly.

KFCA02404-E (R)

`parameter error, func=`*aa...aa*`, item=`*bb...bb*

The function cannot be executed because of an argument specification error.

aa...aa

Name of the function in which the error occurred (If the `_s` version of the function was executed, the corresponding non-`_s` version function name might be output.)

bb...bb

Argument with an error

S: Returns the requested function as an error.

O: Contact the TP1/Client system administrator.

Countermeasure:

Correct the argument error.

KFCA02406-E (R)

`error occurred in TP1/Client, reason=`*aa...aa*`, inf=`*bb...bb*

An error occurred during internal processing of TP1/Client.

aa...aa

Reason for the error

`MEMORY`

Insufficient memory

`RESOURCE`

Resource shortage

`NETWORK`

Network failure

`UNEXPECT`

Unexpected error

`NO SUCH SERVICE`

An illegal request code for the server

`TIMER`

Unsuccessful timer setting

bb...bb

Maintenance information

S: The system stops processing.

O: Record the error log to which this message was output, and then contact the TP1/Client system administrator.

Countermeasure:

Examine and remove the cause of the error according to the output error log. Note that `reason=UNEXPECT` is also output when you specify a port number of the schedule service for an operand for which you must specify a port number of the name service. In this case, check the port number.

KFCA02407-E (R+E)

`error occurred in TP1/Client, reason=`*aa...aa*

An error occurred in TP1/Client.

aa...aa

Reason for the error

One of the following is displayed.

`cltin not executed`

`dc_clt_cltin` function has not been issued.

`cltin already executed`

`dc_clt_cltin` function has already been issued.

`host name`

Host name specification is incorrect.

`TP1/Server not up`

OpenTP1 is inactive at the target node.

S: Stops processing.

O: Check the error log to which this message was output, and examine and remove the cause of the error. Then restart CUP.

KFCA02410-E (R)

`TP1/Server replied error, code=`*aa...aa*`, IPaddr=`*bbbbbbbb*`, port=`*cc...cc*`, inf=`*dd...dd*

An error reply was received from TP1/Server Base.

aa...aa

Error code returned from TP1/Server Base

bbbbbbbb

Sender IP address (hexadecimal number)

cc...cc

Sender port number

dd...dd

Maintenance information

S: Performs one of the following actions:

- Ends the processing that is being executed.
- Ignores the error reply and waits for a reply message. If no reply message arrives after the maximum response wait time in the client environment definition expires, the system returns an error with `DCRPCER_TIMED_OUT`.
- Switches the TP1/Server that is used as a gateway and continues processing, when `Y` is specified for `DCHOSTCHANGE` in the client environment definition.

O: Hand in the error log for this message to the TP1/Client system administrator.

Countermeasure:

Determine the sender process from the sender IP address and the sender port number. Check whether an error has occurred in TP1/Server.

KFCA02411-E (R)

`invalid message received, IPaddr=`*aaaaaaaa*`, port=`*bb...bb*`, inf=`*cc...cc*

A message that cannot be analyzed was received.

aaaaaaaa

Sender IP address (hexadecimal number)

bb...bb

Sender port number

cc...cc

Maintenance information

S: Ignores the received message and waits for the next message.

O: Contact the TP1/Client system administrator.

Countermeasure:

Determine the sender process from the sender IP address and port number, and check the network environment. This message also appears when a function receives a response message that the previous OpenTP1 function could not receive due to a timeout. In this case, ignore the message.

KFCA02412-W (R)

`error-reply received, code=`*aa...aa*`, IPaddr=`*bb...bb*`, port=`*cc...cc*`,`
`inf=`*dd...dd*

An error response was received from TP1/Server.

aa...aa

Error code returned by TP1/Server

bb...bb

IP address of the sending source (in decimal dot notation)

cc...cc

Port number of the sending source

dd...dd

Maintenance information

S: Performs one of the following actions:

- Ends the processing that is being executed.
- Updates the cache information and retries communication.
- Switches the TP1/Server that is used as a gateway and continues processing, when `Y` is specified for `DCHOSTCHANGE` in the client environment definition.

O: Record the error log containing this message and contact the TP1/Client system administrator.

Countermeasure:

Determine the sender process from the sender IP address and the sender port number. Check whether an error has occurred in TP1/Server.

KFCA02419-W (R)

`error occurred in data compression process, group=`*aa...aa*`,`
`service=`*bb...bb*`, reason=`*cc...cc*

An error occurred during internal processing of data compression.

aa...aa

Name of the service group that is requested to perform service

bb...bb

Name of the requested service

cc...cc

Cause of the error

`VERSION`

Version mismatch

`MEMORY`

Insufficient memory

`NOEFFECT`

The data compression is ineffective.

`UNEXPECT`

Unexpected error

S: Requests the service without compressing the data.

O: If the cause of the error is version mismatch, check the version of TP1/Server Base that is requested to perform service and see if it can perform data compression (version 03-03 or later).

If the cause is insufficient memory, follow the instruction in the message that is output before this message.

If the cause is ineffective data compression, check if this message has been output elsewhere in the same CUP since the data becomes larger after compression. Then re-determine whether to use the data compression for each CUP.

If the cause is an unexpected error, contact the TP1/Client system administrator.

Countermeasure:

Examine the cause of the error. If necessary, contact maintenance personnel.

KFCA02420-E (R)

`protocol error, IPaddr=`*aaaaaaaa*`, port=`*bb...bb*`, reason=`*cc...cc*

Incorrect information was detected in the header of the TACT/KERNEL inter-process communication protocol.

aaaaaaaa

Sender IP address (hexadecimal number)

bb...bb

Sender port number

cc...cc

Reason

`SIZE OVER`

A message more than 12 bytes was received as the TACT header.

`SIZE SHORT`

A message less than 12 bytes was received as the TACT header.

`INF2 ERROR`

The contents of additional information 2 are incorrect.

S: Closes the connection and waits for the other system to re-establish a connection. Alternatively, cancels the processing that is being executed and returns the function with an error.

O: Contact the TP1/Client system administrator.

Countermeasure:

When this message is output, the following causes are likely:

- An invalid message was received.
- When *cc....cc* is `SIZE SHORT`, a FIN packet or a RESET packet was received.
- When *cc....cc* is `SIZE SHORT`, a TACT header is divided when it was sent to the TCP buffer.

Determine the sender process from the sender IP address and port number, and check the network environment.

KFCA02421-E (R)

`invalid size, received size=`*aa...aa*`, size in head=`*bb...bb*`, IPaddr=`*cccccccc*`, port=`*dd...dd*

Processing is canceled because the length of the data that reports completion of reception is less than the data length in the header of the TACT/KERNEL inter-process communication protocol.

aa...aa

Received data length

bb...bb

Data length in the header

cccccccc

Sender IP address (hexadecimal number)

dd...dd

Sender port number

S: Cancels processing.

O: Contact the TP1/Client system administrator.

Countermeasure:

Determine the sender process from the sender IP address and the sender port number. Check whether an error has occurred in TP1/Server.

KFCA02431-E (R)

`servicegroup list error, file=`*aa...aa*`:`*bb...bb*`, reason=`*cc...cc*

The file specified with `DCCLTSERVICEGROUPLIST` in the client environment definition contains an error.

aa...aa

File specified with `DCCLTSERVICEGROUPLIST` in the client environment definition

bb...bb

Number of the line that contains an error

cc...cc

`NO VALUE`: No value is specified.

`OUT OF RANGE`: The value is out of the specifiable range.

`INVALID CHAR`: The value contains a character that cannot be specified.

S: Ignores the faulty line, and continues processing.

O: Contact the TP1/Client system administrator.

Countermeasure:

Correct the faulty line.

KFCA02444-E (R)

`communication error, func=`*aa...aa*`, errno=`*bb...bb*

A communication error occurred.

aa...aa

Socket interface function with an error

bb...bb

Error number

S: Stops processing.

O: Contact the TP1/Client system administrator.

Countermeasure:

Examine the cause of the error referring to the function name and error number.

KFCA02445-E (R)

`host resolution failed, func=`*aa...aa*`, errno=`*bb...bb*`, data=`*cc...cc*

Host resolution failed.

aa...aa

Name of a socket interface function in which an error occurred

Either `gethostbyname` or `gethostbyaddr` is output here.

bb...bb

Error number

cc...cc

Host name or IP address from which the host could not be resolved (in decimal dot notation)

S: The function either stops processing and returns an error, or ignores the error and continues processing.

O: Contact the TP1/Client system administrator.

Countermeasure:

Make sure that the specified host name or IP address is correct. If DNS is used for host name resolution, check whether there is an error on the network connected to the DNS server or on the DNS server itself.

KFCA02446-E (R)

`communication error occurred while` *aa...aa* `function was being executed, func=`*bb...bb*`, errno=`*cc...cc*`, c-info=`*dd...dd*`:`*ee...ee*`, s-info=`*ff...ff*`:`*gg...gg*

A communication error occurred.

aa...aa

Name of the executed function

bb...bb

Name of the socket interface function for which the error occurred

cc...cc

Error number

dd...dd

Local IP address (in decimal dot notation)

ee...ee

Local port

ff...ff

Remote IP address (in decimal dot notation)

gg...gg

Remote port

S: Cancels processing.

O: Contact the TP1/Client system administrator.

Countermeasure:

Determine the cause of the error from the information in this message and the applicable data on the server.

KFCA02447-E (R)

`timeout occurred while` *aa...aa* `function was being executed, sts=`*bb...bb*`, time=`*cc...cc*`, c-info=`*dd...dd*`:`*ee...ee*`, s-info=`*ff...ff*`:`*gg...gg*

A timeout occurred.

aa...aa

Name of the executed function

bb...bb

Wait status of the function before the timeout occurred:

`CONNECT`: The function was waiting for a response for connection establishment.

`ACCEPT`: The function was waiting for a connection establishment request.

`RECV(T)`: The function was waiting for the TACT header to be received.

`RECV(D)`: The function was waiting for data to be received.

`RECV(D2)`: The function was waiting for data to be received.

`RECV(D3)`: The function was waiting for data to be received.

`RECV(M)`: The function was waiting for message information to be received.

`RECV(R)`: The function was waiting for response-only data to be received.

`RECVFROM`: The function was waiting for broadcast data to be received.

cc...cc

Timeout value specified in the client environment definition or as an argument of the function (seconds)

dd...dd

Local IP address (in decimal dot notation)

ee...ee

Local port

ff...ff

Remote IP address (in decimal dot notation)

gg...gg

Remote port

S: Cancels processing.

O: Contact the TP1/Client system administrator.

Countermeasure:

Check whether the specified timeout value is appropriate. If the value is appropriate, server processing might have been delayed for a reason such as a line failure. Determine the cause of the error from the information in this message and the applicable data on the server.

KFCA02449-E (R)

`communication error, func=`*aa...aa*`, errno=`*bb...bb*`, remote node addr=`*cc...cc*`, remote port=`*dd...dd*

A communication error occurred.

aa...aa

Function name of the socket interface that caused the error.

bb...bb

Error number

cc...cc

Send destination IP address

dd...dd

Send destination port number

S: Stops processing.

O: Contact the TP1/Client system administrator.

Countermeasure:

Examine the cause of the error referring to the function name and error number.

KFCA02450-W (R)

`The client-ID of a different thread has been specified. func=`*aa...aa*`, thread1=`*bb...bb*`, thread2=`*cc...cc*

The client-ID of a different thread has been specified.

This message is output only once after execution of the `dc_clt_cltin_s` function.

aa...aa

Name of the client function

bb...bb

ID of the thread that started the client function *cc...cc*

ID of the thread that acquired the client ID

S: Continues processing.

O: Record the error log containing this message and contact the TP1/Client system administrator.

Countermeasure:

The same client ID is used across threads. Correct the CUP to ensure that client IDs are specified correctly when functions are executed.

KFCA02451-W (R)

`The function was executed twice.`
`func1=`*aa...aa*`, thread1=`*bb...bb*`, func2=`*cc...cc*`, thread2=`*dd...dd*

A duplicate client functions was executed.

This message is output only once after execution of the `dc_clt_cltin_s` function.

aa...aa

Name of the client function for which duplication was detected

bb...bb

ID of the thread that started the function for which duplication was detected

cc...cc

Name of the client function already being executed

dd...dd

ID of the thread that started the function already being executed

S: Continues processing.

O: Record the error log containing this message and contact the TP1/Client system administrator.

Countermeasure:

The error occurred because client functions with the same client ID were executed concurrently. Correct the CUP to ensure that duplicate functions are not executed.

KFCA02460-I (E+S)

`usage: `*aaaaaaa* `[`*bb* `pathname]`

This message shows the format of the trace edit and output command. This message appears if the specification of the trace edit and output command is incorrect.

aaaaaaa

UAP trace edit and output command

bb

Option

TP1/Client/W

`usage: cltdump [-u|-s|-m] [-n] [-f` *pathname*`]`

TP1/Client/P

`usage: cltdmp32 [/u|/s|/m] [/n] [/f` *pathname*`]`

S: Stops processing.

O: Reenter the command in the correct format.

KFCA02461-E (E)

`file(`*aa...aa*`)not found`

The specified file was not found upon input of the trace edit and output command.

aa...aa

File name specified by the user

S: Stops processing.

O: Correct the file name and then reenter the trace edit and output command.

KFCA02462-E (E)

`no file to edit:` *aa...aa*

There is no trace file to edit and output.

aa...aa

Type of trace file to edit and output.

`UAP`: UAP trace

`SOC`: Socket trace

`MDL`: Module trace

S: Stops processing.

O: If the CUP run-time trace file has not been created, the client environment definition may have an error. Review the specification according to the string indicated in *aa...aa*:

If `UAP` is output, see the `DCTRCUAP` specification.

If `SOC` is output, see the `DCTRCSOC` specification.

If `MDL` is output, see the `DCTRCMDL` specification.

If TP1/Client/W is used, see the `DCTRCPATH` specification.

If the specification is incorrect, correct the error, and then re-execute the CUP.

KFCA02463-E (E)

`error occurred, file=`*aa...aa*

An error occurred while executing the trace edit and output command.

aa...aa

File with an error

S: Stops processing.

O: Contact the TP1/Client system administrator.

Countermeasure:

Examine the cause of the error. If necessary, contact maintenance personnel.

KFCA02466-I (R)

`TP1/Server Base ready, hostname=`*aa...aa*`, port=`*bb...bb*

The TP1/Server (host that the name service queries) to be used as a gateway is determined. Note that the gateway TP1/Server may change depending on the contents of the error.

aa...aa

Host name of the TP1/Server that is used as a gateway

bb...bb

Port number of the TP1/Server that is used as a gateway (port number of the name service)

KFCA02468-I (R)

`TP1/Client changed connected TP1/Server Base, hostname=`*aa...aa*`, port=`*bb...bb*

Because an error occurred during communication with the TP1/Server (host that the name service queries) used as a gateway, the gateway TP1/Server was changed.

aa...aa

Host name of the TP1/Server used as a gateway after switchover

bb...bb

Port number of the TP1/Server that is used as a gateway (port number of the name service) after switchover

KFCA02470-E (R)

`the error commitment for transaction(`*aaaaaaaabbbbbbbb*`) occurred, return value=`*cc...cc*

An attempt was made to execute the `dc_rpc_close_s`, `dc_clt_cltout_s` function without commit or rollback in unchained mode. TP1/Client then automatically performed commit in unchained mode and caused an error in synchronous point processing.

aaaaaaaa

OpenTP1 system node ID (eight letters)

bbbbbbbb

Global transaction number (eight hexadecimal characters)

cc...cc

Return value (4-digit positive number) for synchronous point processing

S: Continues processing.

O: Contact the TP1/Client system administrator.

Countermeasure:

1. Take action against a synchronous point processing error.
2. After issuing a request for commit or rollback in unchained mode, terminate the CUP.

KFCA02471-W (R)

`Current status is outside a transaction. trngid=`*aa...aa*`,`
`trnbid=`*bb...bb*

The current transaction status is out of range. The connection to the transaction executing process has already been released because of an error. Therefore, the `dc_trn_unchained_commit_s` or `dc_trn_unchained_rollback_s` function returned a `DCCLTER_OLTF_NOT_UP` error.

aa...aa

Transaction global identifier of the transaction where an error occurred

bb...bb

Transaction branch identifier of the transaction branch where an error occurred

S: The executed function returns a `DCCLTER_OLTF_NOT_UP` error. Currently, TP1/Client is placed outside a transaction.

O: Contact the TP1/Client system administrator.

Countermeasure:

Find the cause of the error from the message output before this one, and correct the error. If the error was due to a timeout, review the definition and settings so that a timeout does not occur at the client before the server.

To restart the transaction, execute the `dc_trn_begin_s` function. Note that the previous transaction indicated by `trngid` and `trnbid` remains active until the transaction inquiry interval expires.

KFCA02472-E (R)

`The function has been issued from an invalid context. func=`*aa...aa*

The function indicated by `func` was called from incorrect context.

The connection to the transaction executing process has already been released because of an error. Therefore, the function indicated by `func` returned a `DCCLTER_OLTF_NOT_UP` or `DCRPCER_OLTF_NOT_UP` error.

aa...aa

Function name

S: The executed function returns a `DCCLTER_OLTF_NOT_UP` or `DCRPCER_OLTF_NOT_UP` error. TP1/Client remains inside a transaction.

O: Contact the TP1/Client system administrator.

Countermeasure:

Find the cause of the error from the message output before this one, and correct the error. If the error was due to a timeout, review the definition and settings so that a timeout does not occur at the client before the server.

To restart the transaction, execute the `dc_trn_unchained_commit_s` or `dc_trn_unchained_rollback_s` function, and then the `dc_trn_begin_s` function.

We recommend that you write a program so that the `dc_trn_unchained_commit_s` or `dc_trn_unchained_rollback_s` function is executed if an error occurs during transaction processing.

KFCA02480-I (R)

`The permanent connection was established.  IPaddr=`*aa...aa*`, port=`*bb...bb*

Permanent connection has been established. The connection is established with a CUP executing process, a RAP-processing listener, or a DCCM3 logical terminal.

aa...aa

IP address of the destination of the established permanent connection

bb...bb

Port number of the destination of the established permanent connection

KFCA02481-I (R)

`The permanent connection was cut off.  IPaddr=`*aa...aa*`, port=`*bb...bb*

Permanent connection has been disconnected. The connection is established with a CUP executing process, a RAP-processing listener, or a DCCM3 logical terminal.

aa...aa

IP address of the destination of the established permanent connection

bb...bb

Port number of the destination of the established permanent connection

KFCA02482-E(R)

`communication error, func=`*aa...aa*`, errno=`*bb...bb*`, port=`*cc...cc*

A communication error occurred.

aa...aa

Function name of the socket interface that incurred the error

bb...bb

Error number

cc...cc

Number of the port being used

S: Terminates processing.

O: Contact the TP1/Client system administrator.

Countermeasure:

Reference an error number to examine the cause of the error.

KFCA02485-E (R)

`invalid message received while` *aa...aa* `function was being executed. expected message=`*bb...bb*`, received message=`*cc...cc*`, c-info=`*dd...dd*`:`*ee...ee*`, s-info=`*ff...ff*`:`*gg...gg*`, action=`*hh...hh*

While the message assembly facility or message delivery confirmation facility was being used, a message with an invalid value set was received.

aa...aa

Name of the executed function

bb...bb

Expected message (hexadecimal characters) (If there is no information to be output, `****` is output.)

cc...cc

Received message (hexadecimal characters) *dd...dd*

Local IP address (decimal dot notation)

ee...ee

Local port

ff...ff

Remote IP address (decimal dot notation)

gg...gg

Remote port

hh...hh

System action

`RETRY`:

The system continues processing, and retries receive processing.

`STOP`:

The system cancels processing. The function returns an error.

O: Contact the TP1/Client system administrator.

Countermeasure:

Depending on the *hh...hh* value, take action as follows.

`RETRY`:

The system has retried receive processing because the message ID of the received message was invalid. Accordingly, the function might have now terminated normally after receiving a valid message. If the function has not terminated normally, see the message output after this one.

`STOP`:

The system has canceled processing because the length or segment information of the received message was incorrect. Review the settings of the remote system. If messages collided (the `dc_clt_assem_send_s` function returned a `DCCLTER_COLLISION_MESSAGE` error), try again if necessary.

KFCA02486-E (R)

`receive buffer overflowed. argument length=`*aa...aa*`, received length=`*bb...bb*`, received message=`*cc...cc*`, c-info=`*dd...dd*`:`*ee...ee*`, s-info=`*ff...ff*`:`*gg...gg*

When the message assembly facility or message delivery confirmation facility was being used, a message could not be received because the buffer area prepared by the CUP was too small.

aa...aa

Value of the `recvleng` argument for the `dc_clt_assem_receive_s` function (decimal number)

bb...bb

Length of the message to be received (decimal number)

cc...cc

Received message information (hexadecimal characters)

dd...dd

Local IP address (decimal dot notation)

ee...ee

Local port

ff...ff

Remote IP address (decimal dot notation)

gg...gg

Remote port

S: Terminates processing. The `dc_clt_assem_receive_s` function returns a `DCCLTER_INF_TOO_BIG` error. The connection with the remote system is released.

O: Record the error log containing this message and contact the TP1/Client system administrator.

Countermeasure

The message length specified in the function argument is less than the message length set in the received message information. Check whether the value specified in the `recvleng` argument of the `dc_clt_assem_receive_s` function is appropriate. Alternatively, review the UAP on the remote system.

Appendixes

A. Code Conversion Specifications

This appendix explains the code conversion specifications that apply when the character code converter is used. The character code converter can only be used with TP1/Client/P.

The following table shows the character code sets supported by TP1/Client/P.

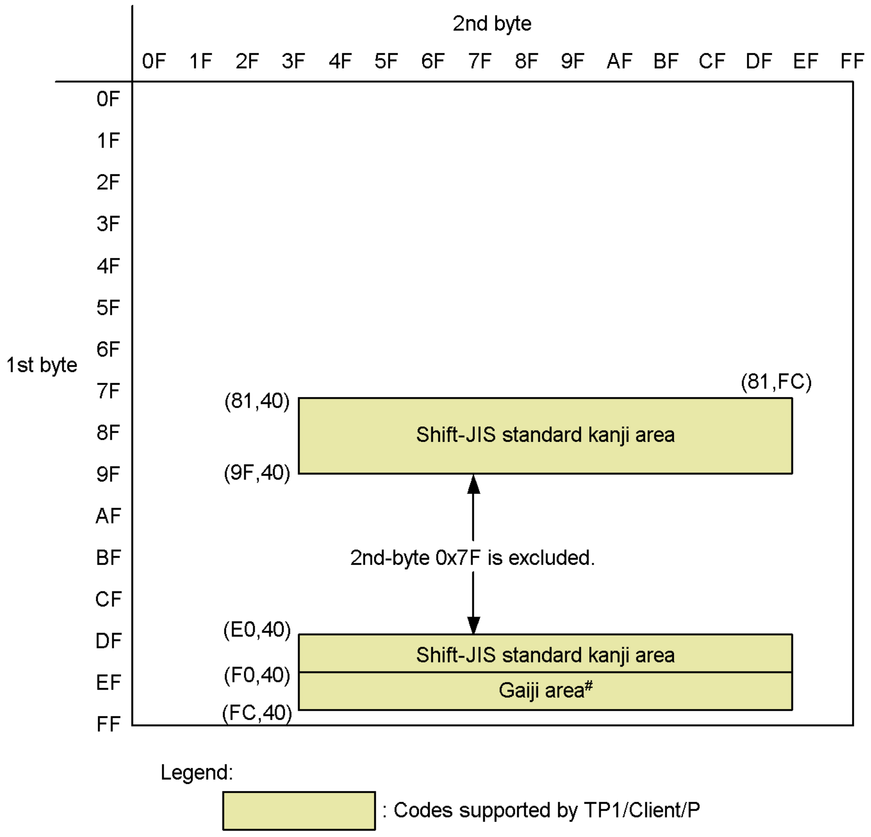
Table A-1: The character code sets supported by TP1/Client/P

Character type	Code set (PC)	Code set (mainframe)
Hankaku	JIS	EBCDIK EBCDIC
Zenkaku	Shift-JIS	KEIS

A.1 Codes supported by TP1/Client/P

This appendix explains the codes supported by TP1/Client/P.

(1) Supported shift-JIS codes

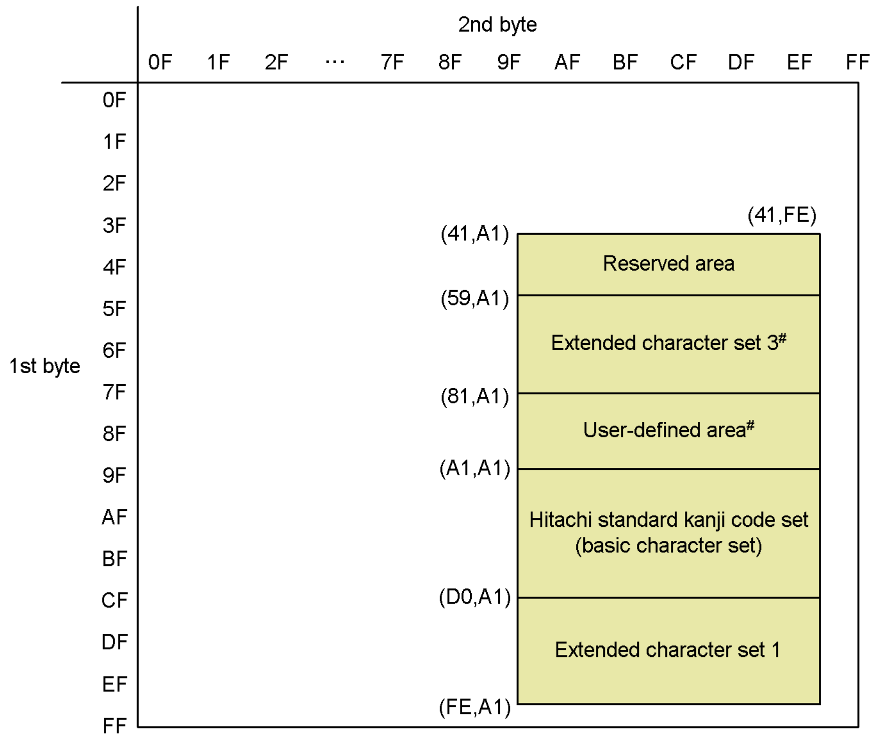

2nd byte
0F 1F 2F 3F 4F 5F 6F 7F 8F 9F AF BF CF DF EF FF
1st byte
0F 1F 2F 3F 4F 5F 6F 7F 8F 9F AF BF CF DF EF FF
(81,40)
(81,FC)
Shift-JIS standard kanji area
(9F,40)
2nd-byte 0x7F is excluded.
(E0,40)
Shift-JIS standard kanji area
(F0,40)
Gaiji area#
(FC,40)
Legend:
: Codes supported by TP1/Client/P

#:

The codes in this area are converted as shift-JIS gaiji codes. For details about the conversion of gaiji codes, see the manual *CommuniNet Version 3*.

Note that the `dc_clt_code_convert` function converts these codes to spaces.

(2) Supported KEIS codes


2nd byte
0F 1F 2F ⋯ 7F 8F 9F AF BF CF DF EF FF
1st byte
0F 1F 2F 3F 4F 5F 6F 7F 8F 9F AF BF CF DF EF FF
(41,A1)
(41,FE)
Reserved area
(59,A1)
Extended character set 3#
(81,A1)
User-defined area#
(A1,A1)
Hitachi standard kanji code set (basic character set)
(D0,A1)
Extended character set 1
(FE,A1)

Legend:
⋯ : Omitted.
▨ : Codes supported by TP1/Client/P.

#:

The codes in this area are converted to shift-JIS gaiji codes. For details about the conversion of gaiji codes, see the manual *CommuniNet Version 3*.

Note that the `dc_clt_code_convert` function converts these codes to spaces.

A.2 Conversion of shift-JIS codes and KEIS codes

The following table explains how shift-JIS codes and KEIS codes are converted.

Table A-2: Specifications for conversion of Shift-JIS and KEIS codes

Shift-JIS code range	KEIS code range	Conversion specifications	
		Shift-JIS to KEIS	**KEIS to shift-JIS**
1st byte: 81 to 9F and E0 to EF 2nd byte: 40 to 7E and 80 to FC	1st byte: A1 to FE 2nd byte: A1 to FE	The Shift-JIS codes are mapped to the KEIS basic character set and extended character set 1.	The KEIS codes are mapped to the shift-JIS standard kanji code area.
1st byte: F0 to FC 2nd byte: 40 to 7E and 80 to FC	1st byte: 41 to A0 2nd byte: A1 to FE	The Shift-JIS codes are mapped to the KEIS gaiji area by using the CommuniNet code mapping utility. For details, see the manual *CommuniNet Version 3*. Note that the `dc_clt_code_convert` function converts these codes to spaces.	The KEIS codes are mapped to the shift-JIS gaiji area by using the CommuniNet code mapping utility. For details, see the manual *CommuniNet Version 3*. Note that the `dc_clt_code_convert` function converts these codes to spaces.

The following table shows conversion from shift-JIS to KEIS83.

Table A-3: Conversion from shift-JIS to KEIS83

L/H	81	82	...	9E	9F	E0	E1	...	EE	EF	F0...FC
00 ... 3F	#1										
40	A1A1	A3A1	...	DBA1	DDA1	DFA1	E1A1	...	FBA1	FDA1	#2
41	A1A2	A3A2	...	DBA2	DDA2	DFA2	E1A2	...	FBA2	FDA2	
...	...	...	...	...	...	...	...	...	...	...	
7E	A1DF	A3DF	...	DBDF	DDDF	DFDF	E1DF	...	FBDF	FDDF	
7F	#1										

L/H	81	82	...	9E	9F	E0	E1	...	EE	EF	F0...FC
80	A1E0	A3E0	...	DBE0	DDE0	DFE0	E1E0	...	FBE0	FDE0	#2
81	A1E1	A3E1	...	DBE1	DDE1	DFE1	E1E1	...	FBE1	FED1	
...	...	...	...	...	...	...	...	...	...	...	
9D	A1FD	A3FD	...	DBFD	DDFD	DFFD	E1FD	...	FBFD	FDFD	
9E	A1FE	A3FE	...	DBFE	DDFE	DFFE	E1FE	...	FBFE	FDFE	
9F	A2A1	A4A1	...	DCA1	DEA1	E0A1	E2A1	...	FBA1	FEA1	
A0	A2A2	A4A2	...	DCA2	DEA2	E0A2	E2A2	...	FBA2	FEA2	
...	...	...	...	...	...	...	...	...	...	...	
FB	A2FD	A4FD	...	DCFD	DEFD	E0FD	E2FD	...	FBFD	FEFD	
FC	A2FE	A4FE	...	DCFE	DEFE	E0FE	E2FE	...	FBFE	FEFE	
FD ...	#1										

Legend:

...: Omitted.

#1:

These codes are assumed to be undefined kanji codes. The `dc_clt_code_convert` and `dc_clt_code_convert_exec` functions convert these codes to spaces or else return an error, depending on the specification of the `flags` argument.

#2:

These codes are converted to shift-JIS gaiji codes.

However, if the `dc_clt_code_convert` function is used, these codes are converted to spaces.

The following table shows conversion from KEIS83 to shift-JIS.

Table A-4: Conversion from KEIS83 to shift-JIS

L/H	00...40	41...A0	A1	A2	...	FD	FE	FF
00	#1	#3						
...								
A0								
A1		#2	8140	819F	...	EF40	EF9F	#3
A2			8141	81A0	...	EF41	EFA0	
A3			8142	81A1	...	EF42	EFA1	
A4			8143	81A2	...	EF43	EFA2	
...			...	...	...	...	...	
DF			817E	81DD	...	EF7E	EFDD	
E0			8180	81DE	...	EF80	EFDE	
E1			8181	81DF	...	EF81	EFDF	
...			...	...	...	...	...	
FE			819E	81FC	...	EF91	EFFC	
FF		#3						

Legend:

...: Omitted.

#1:

These codes are converted as one-byte control codes in EBCDIK and EBCDIC.

#2:

These codes are converted to shift-JIS gaiji codes.

However, if the `dc_clt_code_convert` function is used, these codes are converted to spaces.

#3:

These codes are assumed to be undefined kanji codes. The `dc_clt_code_convert` and `dc_clt_code_convert_exec` functions convert these codes to spaces or else return an error, depending on the specification of the `flags` argument.

The following tables show the code mappings between shift-JIS, KEIS78, and

KEIS83.

Table A-5: Code mappings between shift-JIS, KEIS78, and KEIS83 (1)

No.	Character	Shift-JIS	KEIS78	KEIS83
1	鰺	E9CB	B0B3	F2CD
2	鯵	88B1	F2CD	B0B3
3	鶯	E9F2	B2A9	F2F4
4	鴬	89A7	F2F4	B2A9
5	蠣	E579	B3C2	E9DA
6	蛎	8A61	E9DA	B3C2
7	攪	9D98	B3C9	D9F8
8	撹	8A68	D9F8	B3C9
9	竈	E27D	B3F6	E3DE
10	竃	8A96	E3DE	B3F6
11	灌	9FF3	B4C3	DEF5
12	潅	8AC1	DEF5	B4C3
13	諫	E67C	B4D2	EBDD
14	諌	8AD0	EBDD	B4D2
15	頸	E8F2	B7DB	F0F4
16	頚	8C7A	F0F4	B7DB
17	礦	E1E6	B9DC	E2E8
18	砿	8D7B	E2E8	B9DC
19	蘂	E541	BCC9	E9A2
20	蕊	8EC7	E9A2	BCC9
21	靱	E8D5	BFD9	F0D7
22	靭	9078	F0D7	BFD9
23	賤	E6CB	C1A8	ECCD
24	賎	9147	ECCD	C1A8
25	壺	9AE2	C4DB	D4E4
26	壷	92D9	D4E4	C4DB
27	礪	E1E8	C5D7	E2EA
28	砺	9376	E2EA	C5D7
29	檮	9E8D	C5EE	DBED
30	梼	938E	DBED	C5EE
31	濤	9FB7	C5F3	DEB9
32	涛	9393	DEB9	C5F3
33	邇	E78E	C6F6	EDEE
34	迩	93F4	EDEE	C6F6
35	蠅	E5A2	C7E8	EAA4
36	蝿	9488	EAA4	C7E8
37	檜	9E77	C9B0	DBD8
38	桧	954F	DBD8	C9B0
39	儘	98D4	CBF9	D0D6
40	侭	9699	D0D6	CBF9
41	藪	E54D	CCF9	E9AE
42	薮	96F7	E9AE	CCF9
43	籠	E2C4	CFB6	E4C6
44	篭	9855	E4C6	CFB6
45	堯#	EA9F	5CC3	F4A1
46	遙#	EAA1	6BA2	F4A3
47	瑤#	EAA2	65A7	F4A4

Table A-6: Code mappings between shift-JIS, KEIS78, and KEIS83 (2)

No.	Character	Shift-JIS	KEIS78	KEIS83
48	槇#	EAA0	61FC	F4A2
49	⊃	81BD	A2BE	A2BF
50	⊂	81BC	A2BF	A2BE
51	┌	84A1	AFA1	A8A3
52	┏	84AC	AFA2	A8AE
			AFA3	
53	┐	84A2	AFA4	A8A4
54	┓	84AD	AFA5	A8AF
			AFA6	
55	└	84A4	AFA7	A8A6
56	┗	84AF	AFA8	A8B1
			AFA9	
57	┘	84A3	AFAA	A8A5
58	┛	84AE	AFAB	A8B0
			AFAC	
59	├	84A5	AFAD	A8A7
60	┣	84B0	AFAE	A8B2
			AFAF	
61	┤	84A7	AFB0	A8A9
62	┫	84B2	AFB1	A8B4
			AFB2	
63	┬	84A6	AFB3	A8A8
64	┳	84B1	AFB4	A8B3
			AFB5	
65	┴	84A8	AFB6	A8AA
66	┻	84B3	AFB7	A8B5
			AFB8	
67	┼	84A9	AFB9	A8AB
68	╋	84B4	AFBA	A8B6
			AFBB	
69	─	849F	AFBC	A8A1
70	━	84AA	AFBD	A8AC
			AFBE	
71	│	84A0	AFBF	A8A2
72	┃	84AB	AFC0	A8AD
			AFC1	
			AFC2	
			AFC3	
			AFC4	

#: In conversion from shift-JIS to KEIS78, the shift-JIS codes are converted to the codes in the KEIS gaiji area (extended set 3 area). In conversion from KEIS78 to shift-JIS, the KEIS78 codes are converted to spaces. However, the codes registered in the gaiji mapping table are converted according to the table. For details about gaiji conversion, see the manual *CommuniNet Version 3*.

The following two tables show code conversion from JIS to EBCDIK.

Table A-7: Code conversion from JIS to EBCDIK (1)

L/H	0	1	2	3	4	5	6	7
0	NUL (00)	DLE (10)	SP (40)	0 (F0)	@ (7C)	P (D7)	‘ (79)	p (76)
1	SOH (01)	DC1 (11)	! (4F)	1 (F1)	A (C1)	Q (D8)	a (59)	q (77)
2	STX (02)	DC2 (12)	" (7F)	2 (F2)	B (C2)	R (D9)	b (62)	r (78)
3	ETX (03)	DC3 (13)	# (7B)	3 (F3)	C (C3)	S (E2)	c (63)	s (80)
4	EOT (37)	DC4 (3C)	$ (E0)	4 (F4)	D (C4)	T (E3)	d (64)	t (8B)
5	ENQ (2D)	NAK (3D)	% (6C)	5 (F5)	E (C5)	U (E4)	e (65)	u (9B)
6	ACK (2E)	SYN (32)	& (50)	6 (F6)	F (C6)	V (E5)	f (66)	v (9C)
7	BEL (2F)	ETB (26)	' (7D)	7 (F7)	G (C7)	W (E6)	g (67)	w (A0)
8	BS (16)	CAN (18)	((4D)	8 (F8)	H (C8)	X (E7)	h (68)	x (AB)
9	HT (05)	EM (19)	) (5D)	9 (F9)	I (C9)	Y (E8)	i (69)	y (B0)
A	NL (15)	SUB (3F)	* (5C)	: (7A)	J (D1)	Z (E9)	j (70)	z (B1)
B	VT (0B)	ESC (27)	+ (4E)	; (5E)	K (D2)	[(4A)	k (71)	{ (C0)
C	FF (0C)	FS (1C)	, (6B)	< (4C)	L (D3)	\ (5B)	l (72)	| (6A)
D	CR (0D)	GS (1D)	- (60)	= (7E)	M (D4)	] (5A)	m (73)	} (D0)
E	SO (0E)	RS (1E)	. (4B)	> (6E)	N (D5)	^ (5F)	n (74)	‾ (A1)
F	SI (0F)	US (1F)	/ (61)	? (6F)	O (D6)	_ (6D)	o (75)	DEL (07)

Table A-8: Code conversion from JIS to EBCDIK (2)

L/H	8	9	A	B	C	D	E	F
0	(space) (20)	–	(space) (57)	ｰ (58)	ﾀ (91)	ﾐ (A5)	–	–
1	–	–	｡ (41)	ｱ (81)	ﾁ (92)	ﾑ (A6)	–	–
2	–	–	｢ (42)	ｲ (82)	ﾂ (93)	ﾒ (A7)	–	–
3	–	–	｣ (43)	ｳ (83)	ﾃ (94)	ﾓ (A8)	–	–
4	–	–	､ (44)	ｴ (84)	ﾄ (95)	ﾔ (A9)	–	–
5	–	–	･ (45)	ｵ (85)	ﾅ (96)	ﾕ (AA)	–	–
6	–	–	ｦ (46)	ｶ (86)	ﾆ (97)	ﾖ (AC)	–	–
7	–	–	ｧ (47)	ｷ (87)	ﾇ (98)	ﾗ (AD)	–	–
8	–	–	ｨ (48)	ｸ (88)	ﾈ (99)	ﾘ (AE)	–	–
9	–	–	ｩ (49)	ｹ (89)	ﾉ (9A)	ﾙ (AF)	–	–
A	–	–	ｪ (51)	ｺ (8A)	ﾊ (9D)	ﾚ (BA)	–	–
B	–	–	ｫ (52)	ｻ (8C)	ﾋ (9E)	ﾛ (BB)	–	–
C	–	–	ｬ (53)	ｼ (8D)	ﾌ (9F)	ﾜ (BC)	–	–
D	–	–	ｭ (54)	ｽ (8E)	ﾍ (A2)	ﾝ (BD)	–	(space) (FD)
E	–	–	ｮ (55)	ｾ (8F)	ﾎ (A3)	ﾞ (BE)	–	(space) (FE)
F	–	–	ｯ (56)	ｿ (90)	ﾏ (A4)	ﾟ (BF)	–	(space) (FF)

Legend:
-: Conversion assumes that the code is the first byte of a shift-JIS code.

The following two tables show code conversion from EBCDIK to JIS.

Table A-9: Code conversion from EBCDIK to JIS (1)

L/H	0	1	2	3	4	5	6	7
0	NUL (00)	DEL (10)	(space) (80)	(space) (90)	SP (20)	& (26)	- (2D)	j (6A)
1	SOH (01)	DC1 (11)	(space) (81)	(space) (91)	｡ (A1)	ｪ (AA)	/ (2F)	k (6B)
2	STX (02)	DC2 (12)	(space) (82)	SYN (16)	｢ (A2)	ｫ (AB)	b (62)	l (6C)
3	ETX (03)	DC3 (13)	(space) (83)	(space) (93)	｣ (A3)	ｬ (AC)	c (63)	m (6D)
4	(space) (9C)	(space) (9D)	(space) (84)	(space) (94)	､ (A4)	ｭ (AD)	d (64)	n (6E)
5	HT (09)	NL (0A)	(space) (85)	(space) (95)	･ (A5)	ｮ (AE)	e (65)	o (6F)
6	(space) (86)	BS (08)	ETB (17)	(space) (96)	ｦ (A6)	ｯ (AF)	f (66)	p (70)
7	BEL (7F)	(space) (87)	ESC (1B)	EOT (04)	ｧ (A7)	(space) (A0)	g (67)	q (71)
8	(space) (97)	CAN (18)	(space) (88)	(space) (98)	ｨ (A8)	ｰ (B0)	h (68)	r (72)
9	(space) (8D)	EM (19)	(space) (89)	(space) (99)	ｩ (A9)	a (61)	i (69)	‘ (60)
A	(space) (8E)#	(space) (92)	(space) (8A)	(space) (9A)	[(5B)	] (5D)	| (7C)	: (3A)
B	VT (0B)	(space) (8F)	(space) (8B)	(space) (9B)	. (2E)	\ (5C)	, (2C)	# (23)
C	FF (0C)	FS (1C)	(space) (8C)	DC4 (14)	< (3C)	* (2A)	% (25)	@ (40)
D	CR (0D)	GS (1D)	ENQ (05)	NAK (15)	((28)	) (29)	_ (5F)	' (27)
E	SO (0E)	RS (1E)	ACK (06)	(space) (9E)	+ (2B)	; (3B)	> (3E)	= (3D)
F	SI (0F)	US (1F)	BEL (07)	SUB (1A)	! (21)	^ (5E)	? (3F)	" (22)

#: Conversion assumes that the code is the first byte of a switch code (0x0A41or 0x0A42).

Table A-10: Code conversion from EBCDIK to JIS (2)

L/H	8	9	A	B	C	D	E	F
0	s (73)	ｿ (BF)	w (77)	y (79)	{ (7B)	} (7D)	$ (24)	0 (30)
1	ｱ (B1)	ﾀ (C0)	‾ (7E)	z (7A)	A (41)	J (4A)	(space) (9F)	1 (31)
2	ｲ (B2)	ﾁ (C1)	^ (CD)	(space) (E0)	B (42)	K (4B)	S (53)	2 (32)
3	ｳ (B3)	ﾂ (C2)	ﾎ (CE)	(space) (E1)	C (43)	L (4C)	T (54)	3 (33)
4	ｴ (B4)	ﾃ (C3)	ﾏ (CF)	(space) (E2)	D (44)	M (4D)	U (55)	4 (34)
5	ｵ (B5)	ﾄ (C4)	ﾐ (D0)	(space) (E3)	E (45)	N (4E)	V (56)	5 (35)
6	ｶ (B6)	ﾅ (C5)	ﾑ (D1)	(space) (E4)	F (46)	O (4F)	W (57)	6 (36)
7	ｷ (B7)	ﾆ (C6)	ﾒ (D2)	(space) (E5)	G (47)	P (50)	X (58)	7 (37)
8	ｸ (B8)	ﾇ (C7)	ﾓ (D3)	(space) (E6)	H (48)	Q (51)	Y (59)	8 (38)
9	ｹ (B9)	ﾈ (C8)	ﾔ (D4)	(space) (E7)	I (49)	R (52)	Z (5A)	9 (39)
A	ｺ (BA)	ﾉ (C9)	ﾕ (D5)	ﾚ (DA)	(space) (E8)	(space) (EE)	(space) (F4)	(space) (FA)
B	t (74)	u (75)	x (78)	ﾛ (DB)	(space) (E9)	(space) (EF)	(space) (F5)	(space) (FB)
C	ｻ (BB)	v (76)	ﾖ (D6)	ﾜ (DC)	(space) (EA)	(space) (F0)	(space) (F6)	(space) (FC)
D	ｼ (BC)	ﾊ (CA)	ﾗ (D7)	ﾝ (DD)	(space) (EB)	(space) (F1)	(space) (F7)	(space) (FD)
E	ｽ (BD)	ﾋ (CB)	ﾘ (D8)	ﾞ (DE)	(space) (EC)	(space) (F2)	(space) (F8)	(space) (FE)
F	ｾ (BE)	ﾌ (CC)	ﾙ (D9)	ﾟ (DF)	(space) (ED)	(space) (F3)	(space) (F9)	E0 (FF)

The following two tables show code conversion from JIS to EBCDIC.

Table A-11: Code conversion from JIS to EBCDIC (1)

L/H	0	1	2	3	4	5	6	7
0	NUL (00)	DLE (10)	SP (40)	0 (F0)	@ (7C)	P (D7)	‘ (79)	p (97)
1	SOH (01)	DC1 (11)	! (4F)	1 (F1)	A (C1)	Q (D8)	a (81)	q (98)
2	STX (02)	DC2 (12)	" (7F)	2 (F2)	B (C2)	R (D9)	b (82)	r (99)
3	ETX (03)	DC3 (13)	# (7B)	3 (F3)	C (C3)	S (E2)	c (83)	s (A2)
4	EOT (37)	DC4 (3C)	$ (E0)	4 (F4)	D (C4)	T (E3)	d (84)	t (A3)
5	ENQ (2D)	NAK (3D)	% (6C)	5 (F5)	E (C5)	U (E4)	e (85)	u (A4)
6	ACK (2E)	SYN (32)	& (50)	6 (F6)	F (C6)	V (E5)	f (86)	v (A5)
7	BEL (2F)	ETB (26)	' (7D)	7 (F7)	G (C7)	W (E6)	g (87)	w (A6)
8	BS (16)	CAN (18)	((4D)	8 (F8)	H (C8)	X (E7)	h (88)	x (A7)
9	HT (05)	EM (19)	) (5D)	9 (F9)	I (C9)	Y (E8)	i (89)	y (A8)
A	NL (15)	SUB (3F)	* (5C)	: (7A)	J (D1)	Z (E9)	j (91)	z (A9)
B	VT (0B)	ESC (27)	+ (4E)	; (5E)	K (D2)	[(4A)	k (92)	{ (C0)
C	FF (0C)	FS (1C)	, (6B)	< (4C)	L (D3)	\ (5B)	l (93)	| (6A)
D	CR (0D)	GS (1D)	- (60)	= (7E)	M (D4)	] (5A)	m (94)	} (D0)
E	SO (0E)	RS (1E)	. (4B)	> (6E)	N (D5)	^ (5F)	n (95)	‾ (A1)
F	SI (0F)	US (1F)	/ (61)	? (6F)	O (D6)	_ (6D)	o (96)	DEL (07)

Table A-12: Code conversion from JIS to EBCDIC (2)

L/H	8	9	A	B	C	D	E	F
0	(20)	−	(57)	(40)	(40)	(40)	−	−
1	−	−	(40)	(40)	(40)	(40)	−	−
2	−	−	(40)	(40)	(40)	(40)	−	−
3	−	−	(40)	(40)	(40)	(40)	−	−
4	−	−	(40)	(40)	(40)	(40)	−	−
5	−	−	(40)	(40)	(40)	(40)	−	−
6	−	−	(40)	(40)	(40)	(40)	−	−
7	−	−	(40)	(40)	(40)	(40)	−	−
8	−	−	(40)	(40)	(40)	(40)	−	−
9	−	−	(40)	(40)	(40)	(40)	−	−
A	−	−	(40)	(40)	(40)	(40)	−	−
B	−	−	(40)	(40)	(40)	(40)	−	−
C	−	−	(40)	(40)	(40)	(40)	−	−
D	−	−	(40)	(40)	(40)	(40)	−	(FD)
E	−	−	(40)	(40)	(40)	(40)	−	(FE)
F	−	−	(40)	(40)	(40)	(40)	−	(FF)

Legend:

-: Conversion assumes that the code is the first byte of a shift-JIS code.

The following two tables show code conversion from EBCDIC to JIS.

Table A-13: Code conversion from EBCDIC to JIS (1)

L/H	0	1	2	3	4	5	6	7
0	NUL (00)	DEL (10)	(space) (80)	(space) (90)	SP (20)	& (26)	- (2D)	(space) (BA)
1	SOH (01)	DC1 (11)	(space) (81)	(space) (91)	‘ (A0)	(space) (A9)	/ (2F)	(space) (BB)
2	STX (02)	DC2 (12)	(space) (82)	SYN (16)	(space) (A1)	(space) (AA)	(space) (B2)	(space) (BC)
3	ETX (03)	DC3 (13)	(space) (83)	(space) (93)	(space) (A2)	(space) (AB)	(space) (B3)	(space) (BD)
4	(space) (9C)	(space) (9D)	(space) (84)	(space) (94)	(space) (A3)	(space) (AC)	(space) (B4)	(space) (BE)
5	HT (09)	NL (0A)	(space) (85)	(space) (95)	(space) (A4)	(space) (AD)	(space) (B5)	(space) (BF)
6	(space) (86)	BS (08)	ETB (17)	(space) (96)	(space) (A5)	(space) (AE)	(space) (B6)	(space) (C0)
7	DEL (7F)	(space) (87)	ESC (1B)	EOT (04)	(space) (A6)	(space) (AF)	(space) (B7)	(space) (C1)
8	(space) (97)	CAN (18)	(space) (88)	(space) (98)	(space) (A7)	(space) (B0)	(space) (B8)	(space) (C2)
9	(space) (8D)	EM (19)	(space) (89)	(space) (99)	(space) (A8)	(space) (B1)	(space) (B9)	' (60)
A	(space) (8E)#	(space) (92)	(space) (8A)	(space) (9A)	[(5B)	] (5D)	| (7C)	: (3A)
B	VT (0B)	(space) (8F)	(space) (8B)	(space) (9B)	. (2E)	\ (5C)	, (2C)	# (23)
C	FF (0C)	FS (1C)	(space) (8C)	DC4 (14)	< (3C)	* (2A)	% (25)	@ (40)
D	CR (0D)	GS (1D)	ENQ (05)	NAK (15)	((28)	) (29)	_ (5F)	' (27)
E	SO (0E)	RS (1E)	ACK (06)	(space) (9E)	+ (2B)	; (3B)	> (3E)	= (3D)
F	SI (0F)	US (1F)	BEL (07)	SUB (1A)	! (21)	^ (5E)	? (3F)	" (22)

#: Conversion assumes that the code is the first byte of a switch code (0x0A41or 0x0A42).

Table A-14: Code conversion from EBCDIC to JIS (2)

L/H	8	9	A	B	C	D	E	F
0	(space) (C3)	(space) (CA)	(space) (D1)	(space) (D8)	{ (7B)	} (7D)	$ (24)	0 (30)
1	a (61)	j (6A)	‾ (7E)	(space) (D9)	A (41)	J (4A)	(space) (20)	1 (31)
2	b (62)	k (6B)	s (73)	(space) (DA)	B (42)	K (4B)	S (53)	2 (32)
3	c (63)	l (6C)	t (74)	(space) (DB)	C (43)	L (4C)	T (54)	3 (33)
4	d (64)	m (6D)	u (75)	(space) (DC)	D (44)	M (4D)	U (55)	4 (34)
5	e (65)	n (6E)	v (76)	(space) (DD)	E (45)	N (4E)	V (56)	5 (35)
6	f (66)	o (6F)	w (77)	(space) (DE)	F (46)	O (4F)	W (57)	6 (36)
7	g (67)	p (70)	x (78)	(space) (DF)	G (47)	P (50)	X (58)	7 (37)
8	h (68)	q (71)	y (79)	(space) (E0)	H (48)	Q (51)	Y (59)	8 (38)
9	i (69)	r (72)	z (7A)	(space) (E1)	I (49)	R (52)	Z (5A)	9 (39)
A	(space) (C4)	(space) (CB)	(space) (D2)	(space) (E2)	(space) (E8)	(space) (EE)	(space) (F4)	(space) (FA)
B	(space) (C5)	(space) (CC)	(space) (D3)	(space) (E3)	(space) (E9)	(space) (EF)	(space) (F5)	(space) (FB)
C	(space) (C6)	(space) (CD)	(space) (D4)	(space) (E4)	(space) (EA)	(space) (F0)	(space) (F6)	(space) (FC)
D	(space) (C7)	(space) (CE)	(space) (D5)	(space) (E5)	(space) (EB)	(space) (F1)	(space) (F7)	(space) (FD)
E	(space) (C8)	(space) (CF)	(space) (D6)	(space) (E6)	(space) (EC)	(space) (F2)	(space) (F8)	(space) (FE)
F	(space) (C9)	(space) (D0)	(space) (D7)	(space) (E7)	(space) (ED)	(space) (F3)	(space) (F9)	E0 (FF)

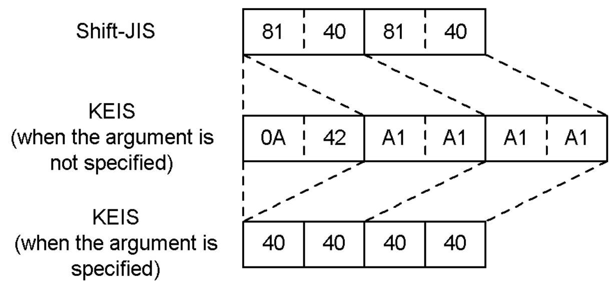
A.3 Code conversion examples

The result of converting some characters is different depending on the value specified for the `flags` argument of the `dc_clt_code_convert` or `dc_clt_codeconv_exec` function. This section shows the differences.

(1) If DCCLT_CNV_SPCHAN is specified

If `DCCLT_CNV_SPCHAN` is specified for the `flags` argument of the `dc_clt_code_convert` or `dc_clt_codeconv_exec` function, a zenkaku space is converted to two hankaku spaces. The following table shows the character code correspondence when `DCCLT_CNV_SPCHAN` is specified.

Figure A-1: Character code correspondence when DCCLT_CNV_SPCHAN is specified


Shift-JIS
81
40
81
40
KEIS
(when the argument is not specified)
0A
42
A1
A1
A1
A1
KEIS
(when the argument is specified)
40
40
40
40

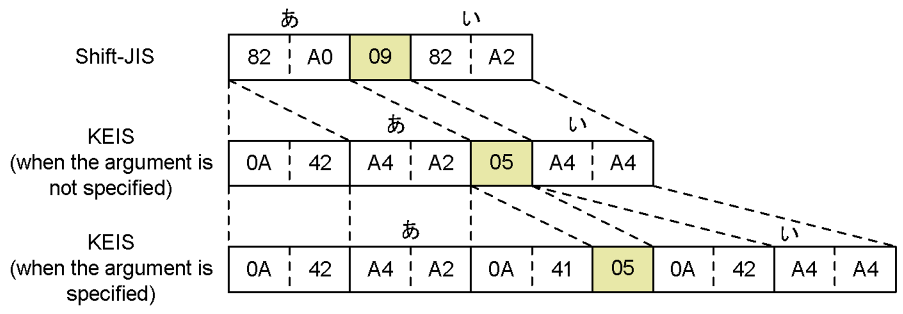
Explanation of codes:

8140: Shift-JIS zenkaku space
0A42: Zenkaku start code
A1A1: KEIS zenkaku space
40: EBCDIK/EBCDIC hankaku space

(2) If DCCLT_CNV_TAB is specified

If `DCCLT_CNV_TAB` is specified for the `flags` argument of the `dc_clt_code_convert` or `dc_clt_codeconv_exec` function, a tab code is converted to the corresponding hankaku code. If a tab code is adjacent to a zenkaku code, shift codes are inserted. The following table shows the character code correspondence when `DCCLT_CNV_TAB` is specified.

Figure A-2: Character code correspondence when DCCLT_CNV_TAB is specified



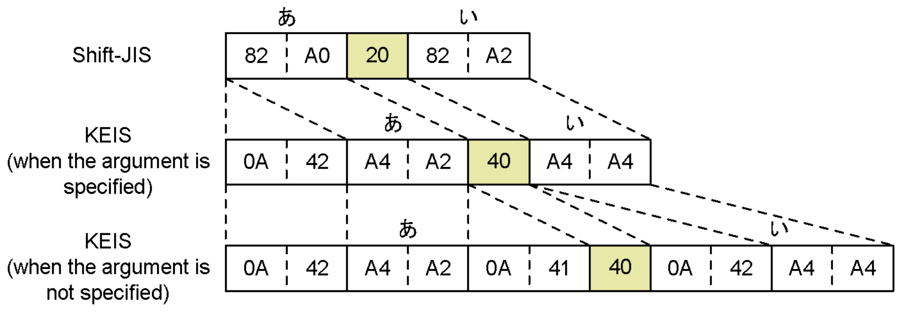
Explanation of codes:

09: JIS tab code
0A42: Zenkaku start code
05: EBCDIK/EBCDIC tab code
0A41: Hankaku start code

(3) If DCCLT_CNV_CNTL is specified

If `DCCLT_CNV_CNTL` is specified for the `flags` argument of the `dc_clt_code_convert` or `dc_clt_codeconv_exec` function, a control code is converted to a hankaku code. If a control code is adjacent to a zenkaku code, shift codes are inserted. The following table shows the character code correspondence when `DCCLT_CNV_CNTL` is specified.

Figure A-3: Character code correspondence when DCCLT_CNV_CNTL is specified


Shift-JIS
あ
い
82
A0
20
82
A2
KEIS (when the argument is specified)
あ
い
0A
42
A4
A2
40
A4
A4
KEIS (when the argument is not specified)
あ
い
0A
42
A4
A2
0A
41
40
0A
42
A4
A4

Explanation of codes:

20: JIS hankaku space
0A42: Zenkaku start code
40: EBCDIK/EBCDIC hankaku space
0A41: Hankaku start code

A.4 Notes on code conversion

- If the input data begins with a KEIS code, add the zenkaku start code (0x0A42).
- If there is no gaiji mapping table, gaiji characters are converted to spaces. For details about the gaiji mapping table, see the manual *CommuniNet Version 3*.
- During code conversion from EBCDIK, EBCDIC, or KEIS to JIS or shift-JIS, a code that does not exist in the conversion table is converted as shown below.
    - If `DCCLT_CNV_INVSPC` is specified for the `flags` argument of the `dc_clt_code_convert` or `dc_clt_codeconv_exec` function, 0x0A is converted to the applicable code unless the byte immediately after 0x0A is either 0x41 or 0x42.
    - If a hankaku code in the range from 0x00 to 0x40 is detected during conversion in zenkaku mode, the code is converted to the applicable one-byte code.

B. Version Changes

This appendix shows the changes made in specific versions by type of modification as follows:

- Addition and deletion of functions, definition operands, and commands
- Operation change
- Change of function, definition operand, and command default values

B.1 Changes made in 07-02

The following table shows the addition and deletion of functions, definition operands, and commands made in TP1/Client/W 07-02 and TP1/Client/P 07-02.

Table B-1: Addition and deletion of functions, definition operands, and commands made in TP1/Client/W 07-02 and TP1/Client/P 07-02

Modification type	Category	Explanation
Addition	Function	`dc_clt_assem_send_s`
		`dc_clt_assem_receive_s`
		`CBLDCCLS('STCONIF ')`
		`CBLDCCLS('ASMSEND ')`
		`CBLDCCLS('ASMRECV ')`
	Operand	`DCCLTDELIVERYCHECK` operand in the client environment definition
		`DCCLTPRFINFOSEND` operand in the client environment definition
		`DCCLTCUPSNDHOST` operand in the client environment definition
	Command	None
Deletion	None	

The following table shows the operation changes made in TP1/Client/W 07-02 and TP1/Client/P 07-02.

Table B-2: Operation changes made in TP1/Client/W 07-02 and TP1/Client/P 07-02

Category	Explanation
Function	None

Category	Explanation
Operand	The range of specifiable values for the `DCSOCKOPEN` operand in the client environment definition has been changed to 1 to 4096.
Command	None
Other category	The message assembly facility has been added.
	The message delivery confirmation facility has been added.
	A facility that allows you to specify the CUP send host has been added.

The following table shows the default-value changes made in TP1/Client/W 07-02 and TP1/Client/P 07-02.

Table B-3: Default-value changes made in TP1/Client/W 07-02 and TP1/Client/P 07-02

Category	Explanation
Function	None
Operand	The default value of the `DCSCDMULTICOUNT` operand in the client environment definition has been changed to 1.
Command	None

B.2 Changes made in 07-01

The following table shows the addition and deletion of functions, definition operands, and commands made in TP1/Client/W 07-01 and TP1/Client/P 07-01.

Table B-4: Addition and deletion of functions, definition operands, and commands made in TP1/Client/W 07-01 and TP1/Client/P 07-01

Modification type	Category	Explanation
Addition	Function	None
	Operand	`DCCLTRECVBUFSIZE` operand in the client environment definition
		`DCCLTSENDBUFSIZE` operand in the client environment definition
		`DCCLTCPNODELAY` operand in the client environment definition
	Command	None
Deletion	None	

No operation changes were made in TP1/Client/W 07-01 and TP1/Client/P 07-01.

Also, no function, definition operand, and command default-value changes were made in TP1/Client/W 07-01 and TP1/Client/P 07-01.

B.3 Changes made in 07-00

The following table shows the addition and deletion of functions, definition operands, and commands made in TP1/Client/W 07-00 and TP1/Client/P 07-00.

Table B-5: Addition and deletion of functions, definition operands, and commands made in TP1/Client/W 07-00 and TP1/Client/P 07-00

Modification type	Category	Explanation
Addition	Function	`CBLDCCLS('EXCLTIN ')`
		`CBLDCCLS('EXSEND  ')`
		`CBLDCCLS('EXNACPT ')`
		`CBLDCCLS('EXNCANCL')`
		`CBLDCCLS('EXNCACPT')`
	Operand	`DCCLTRPCMAXMSGSIZE` operand in the client environment definition
	Command	None
Deletion	None	

The following table shows the operation changes made in TP1/Client/W 07-00 and TP1/Client/P 07-00.

Table B-6: Operation changes made in TP1/Client/W 07-00 and TP1/Client/P 07-00

Category	Explanation
Function	The maximum length of the host name specifiable for or returnable to an argument of the following functions by using the `DCCLTOPTION` operand in the client environment definition was changed from 63 characters to 255 characters: • `dc_clt_cltin_s` • `DCRPC_DIRECT_SCHEDULE` • `dc_clt_set_raphost_s` • `dc_clt_get_raphost_s` • `dc_clt_send_s` • `dc_clt_accept_notification_s` • `dc_clt_cancel_notification_s` • `dc_clt_chained_accept_notification_s` • `CBLDCCLS('STRAPHST')` • `CBLDCCLS('GTRAPHST')`

Category	Explanation
	The maximum length of user data specifiable for or returnable to an argument of the following functions was changed from 1 MB to the `DCCLTRPCMAXMSGSIZE` value in the client environment definition: • `dc_rpc_call_s` • `dc_rpc_call_to_s` • `dc_clt_accept_notification_s` • `dc_clt_cancel_notification_s` • `dc_clt_chained_accept_notification_s` • `CBLDCRPS('CALL ')` • `CBLDCCLS('NOTIFY ')` • `CBLDCCLS('EXNACPT ')` • `CBLDCCLS('CANCEL ')` • `CBLDCCLS('EXNCANCL')` • `CBLDCCLS('A-NOTIFY')` • `CBLDCCLS('EXNCACPT')`
	Operation was changed so that the host name was not stored when NULL was specified for the `hostname` argument of the following functions: • `dc_clt_accept_notification_s` • `dc_clt_chained_accept_notification_s`
Operand	For the `DCCLTOPTION` operand in the client environment definition, the `00000008` specification, which extends the maximum host name length, was added.
	The maximum length of the host name specifiable for any of the following operands in the client environment definition by using the `DCCLTOPTION` operand in the client environment definition was changed from 63 characters to 255 characters: • `DCHOST` • `DCCLTRAPHOST` • `DCCLTDCCMHOST` • `DCSNDHOST` • `DCCLTSERVICEGROUPLIST`
Command	None
Other category	A facility that extends the maximum host name length was added.

No function, definition operand, and command default-value changes were made in TP1/Client/W 07-00 and TP1/Client/P 07-00.

Index

Reader's Comment Form

We would appreciate your comments and suggestions on this manual. We will use these comments to improve our manuals. When you send a comment or suggestion, please include the manual name and manual number. You can send your comments by any of the following methods:

- Send email to your local Hitachi representative.
- Send email to the following address:
  WWW-mk@itg.hitachi.co.jp
- If you do not have access to email, please fill out the following information and submit this form to your Hitachi representative:

Manual name:	
Manual number:	
Your name:	
Company or organization:	
Street address:	
Comment:	

(For Hitachi use)