OpenTP1 Version 7

TP1/Client User's Guide TP1/Client/J

3000-3-D59-20(E)

■ Relevant program product

P-2464-7394 uCosminexus TP1/Client/J 07-02

P-2464-73A4 uCosminexus TP1/Client/J 07-02

This manual can be used for other products, in addition to the product 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-D59(E)): October 2006

Edition 2 (3000-3-D59-20(E)): March 2009

Summary of amendments

The following table lists changes in this manual (3000-3-D59-20(E)) and the product changes related to uCosminexus TP1/Client/J 07-02.

Changes	Location
Parallel processing of receipt of service request messages is now supported in order to prevent scheduling delays (multi-scheduler facility). Due to this change, the following operands have been added to the TP1/Client/J environment definition: • `dcscdmulti` • `dcscdmulticount`	2.2.13, 5.2.1, 5.2.2
An operation similar to batch terminal startup provided by conventional mainframe OLTP is now supported (unidirectional server message reception function). Due to this change, the following methods have been added to the `TP1Client` class: • `acceptNotification` • `cancelNotification` • `openNotification` • `acceptNotificationChained` • `closeNotification` In addition, the following classes have been added: • `ErrAcceptCanceledException` • `ErrVersionException`	2.5, 2.11.2, 3.1, 4. Class TP1Client, 4. Class ErrAcceptCanceledException, 4. Class ErrVersionException
UAP traces are now acquired from the following methods of the `TP1Client` class: • `setUapTraceMode` • `setErrorTraceMode` • `setMethodTraceMode` • `setDataTraceMode`	2.11.2
Performance analysis trace identification information is now output to UAP traces.	2.11.2, 2.11.7(5)
When an RPC is issued to TP1/Server version 07-02 or later using the remote API facility, identification information such as an IP address can now be added to the RPC message that is sent to the scheduler.	2.11.7(2)
When connection establishment is requested by an RPC or TCP/IP communication, the source host can now be specified (source host specification facility). Due to this change, the `dccltcupsndhost` operand has been added to the TP1/Client/J environment definition.	2.13, 5.2.1, 5.2.2

Changes	Location
The receive port can now be fixed for RPCs that use the scheduler direct facility and the name service (receive port fixing facility). Due to this change, the `dccltcuprcvport` operand has been added to the TP1/Client/J environment definition.	2.14, 5.2.1, 5.2.2
A list of the TP1/Client/J environment definition operands has been added.	5.1.1
The maximum execution time from startup to termination of a transaction branch that is executed by the RAP-processing server can now be specified. Due to this change, the `dcclttrcmplmttm` operand has been added to the TP1/Client/J environment definition.	5.2.1, 5.2.2
Notes have been added about the `dcrcvport` operand of the TP1/Client/J environment definition.	5.2.2
Changes to the API, definitions, commands, and default values during upgrading have been described.	Appendix A

The following table lists changes in this manual (3000-3-D59-20(E)) and the product changes related to uCosminexus TP1/Client/J 07-01.

Changes	Location
Information has been added about the files provided by DTP1/Client/J and the directories for storing the files.	1.3.1(1)
A facility for setting terminal identification information has been added in order to fix a DCCM3 logical terminal that is allocated to a CUP when communication is established with a DCCM3 logical terminal by means of a permanent connection. Due to this change, the `setConnectInformation` method has been added to the `TP1Client` class. In addition, the `dccltconnectinf` operand has been added to the TP1/Client/J environment definition.	2.9.1(6), 2.9.3, Table 2-29, Table 3-2, 4. Class TP1Client, 5.2.1, 5.2.2
A data compression facility has been added in order to compress user data that is sent over the network by RPC. Due to this change, the `dccltdatacomp` operand has been added to the TP1/Client/J environment definition.	Table 2-42, 2.12, 5.2.1, 5.2.2
Notes have been added about the execution of methods of instances of the same `TP1Client` class.	3.2

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

Preface

This manual describes the functionality and usage of the following program products:

- P-2464-7394 uCosminexus TP1/Client/J
- P-2464-73A4 uCosminexus TP1/Client/J

These products can be used on operating systems that support Java(TM)2 Software Development Kit, Standard Edition, Version 5.0 or later.

For details about products described in this manual other than the above program product, check the release schedule for the OpenTP1 Version 7 compliant version of the product in question.

Intended readers

This manual is intended for system designers, programmers, and operators, as well as system administrators.

Organization of this manual

This manual consists of the following chapters and an appendix:

1. Overview

Chapter 1 provides a functional overview of TP1/Client/J and describes its features.

2. Functionality

Chapter 2 describes the functionality of TP1/Client/J.

3. Program Interface

Chapter 3 describes the program interface for using TP1/Client/J.

4. Classes Used with TP1/Client/J

Chapter 4 describes the classes that are used with TP1/Client/J.

5. Definitions

Chapter 5 describes the TP1/Client/J environment definitions.

6. Error Handling

Chapter 6 describes the procedures for handling errors.

Appendix A. Changes During Upgrading

Appendix A describes the changes to APIs, definitions, and commands from version to version.

Related publications

This manual is part of a related set of manuals. The manuals in the set, including this manual, are listed below (with the manual numbers):

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 TP1/LiNK User's Guide* (3000-3-D60(E))#1
- *OpenTP1 Version 7 Protocol TP1/NET/TCP/IP* (3000-3-D70(E))
- *OpenTP1 Version 7 TP1/Message Queue User's Guide* (3000-3-D90(E))#1
- *OpenTP1 Version 7 TP1/Message Queue Messages* (3000-3-D91(E))#1
- *OpenTP1 Version 7 TP1/Message Queue Application Programming Guide* (3000-3-D92(E))#1
- *OpenTP1 Version 7 TP1/Message Queue Application Programming Reference* (3000-3-D93(E))#1

Other OpenTP1 products

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

Other related products

- *VOS3 Data Management System XDM 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 or meaning	Abbreviation	
uCosminexus Application Server Enterprise	Cosminexus Application Server	
uCosminexus Application Server Standard		
VOS1 DCCM3	DCCM3	
VOS3 XDM/DCCM3		
Java(TM)	Java	
Java(TM)2 Software Development Kit, Standard Edition	JDK	
uCosminexus TP1/Client/J	TP1/Client/J	
uCosminexus TP1/Client/P	TP1/Client/P	
uCosminexus TP1/Client/W	TP1/Client/W	
uCosminexus TP1/NET/TCP/IP	TP1/NET/TCP/IP	
uCosminexus TP1/LiNK	TP1/LiNK	TP1/Server
uCosminexus TP1/Server Base	TP1/Server Base	
uCosminexus TP1/Web	TP1/Web	
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	

Full name or meaning	Abbreviation	
Microsoft$^{(R)}$ Windows Server$^{(R)}$ 2003, Datacenter Edition	Windows Server 2003 (32bit)	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 Server$^{(R)}$ 2003, Enterprise Edition for Itanium-based systems	Windows Server 2003 (64bit)	
Microsoft$^{(R)}$ Windows Vista$^{(R)}$ Business (x86)	Windows Vista(32bit)	Windows Vista
Microsoft$^{(R)}$ Windows Vista$^{(R)}$ Enterprise (x86)		
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 Professional Operating System	Windows XP	

- Unless functional differences exist, Windows Server 2003, Windows XP and Windows Vista are referred to collectively as *Windows*.

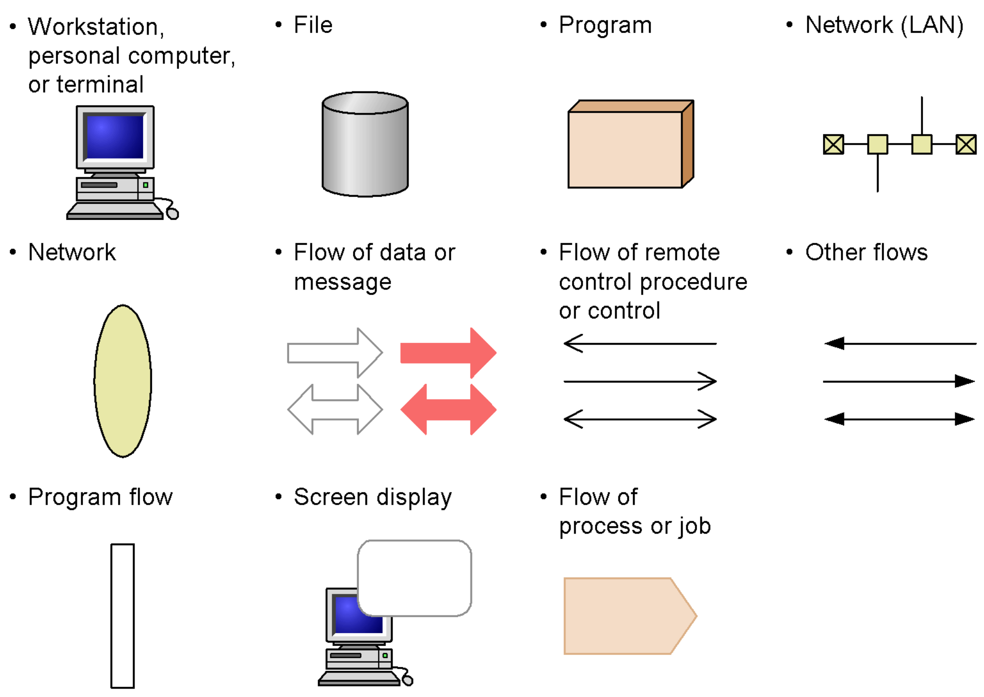
This manual also uses the following abbreviations:

Abbreviation	Full name or meaning
API	Application Programming Interface

Abbreviation	Full name or meaning
CGI	Common Gateway Interface
CUP	Client User Program
DCE	Distributed Computing Environment
EJB	Enterprise Java Beans(TM)
FD	Floppy Disk
HTTP	Hyper Text Transfer Protocol
J2EE	Java 2 Enterprise Edition
JDK	Java(TM) Development Kit
JSP	JavaServer Pages(TM)
MHP	Message Handling Program
OLTP	Online Transaction Processing
OS	Operating System
PC	Personal Computer
PRF	PeRFormance
RAP	Remote Application Programming Interface
RPC	Remote Procedure Call
SPP	Service Providing Program
TCP/IP	Transmission Control Protocol / Internet Protocol
TMS-4V/SP	Transaction Management System-4V/System Product
UAP	User Application Program
VM	Virtual Machine
WS	Workstation
WWW	World Wide Web

Conventions: Diagrams

This manual uses the following conventions in diagrams:


• Workstation, personal computer, or terminal
• File
• Program
• Network (LAN)
• Network
• Flow of data or message
• Flow of remote control procedure or control
• Other flows
• Program flow
• Screen display
• Flow of process or job

Conventions: Fonts and symbols

Font and symbol conventions are classified as:

- General font conventions
- Conventions in syntax explanations

These conventions are described below.

General font conventions

The following table lists the general font conventions:

Font	Convention
Bold	Bold type indicates text on a window, other than the window title. Such text includes menus, menu options, buttons, radio box options, or explanatory labels. For example, bold is used in sentences such as the following: • From the **File** menu, choose **Open**. • Click the **Cancel** button. • In the **Enter name** entry box, type your name.

Font	Convention
Italics	Italics are used to indicate a placeholder for some actual text provided by the user or system. Italics are also used for emphasis. For example: • Write the command as follows: `copy` *source-file target-file* • Do *not* delete the configuration file.
`Code font`	A code font indicates text that the user enters without change, or text (such as messages) output by the system. For example: • At the prompt, enter `dir`. • Use the `send` command to send mail. • The following message is displayed: `The password is incorrect.`

Examples of coding and messages appear as follows (although there may be some exceptions, such as when coding is included in a diagram):

```
MakeDatabase
...
StoreDatabase temp DB32
```

In examples of coding, an ellipsis (...) indicates that one or more lines of coding are not shown for purposes of brevity.

Conventions in syntax explanations

Syntax definitions appear as follows:

`S``tore`**`D`**`atabase [temp|`<u>`perm`</u>`] (`*database-name* `...)`

The following table lists the conventions used in syntax explanations:

Example font or symbol	Convention
`StoreDatabase`	Code-font characters must be entered exactly as shown.
database-name	This font style marks a placeholder that indicates where appropriate characters are to be entered in an actual command.
`SD`	Bold code-font characters indicate the abbreviation for a command.
<u>`perm`</u>	Underlined characters indicate the default value.
`[ ]`	Square brackets enclose an item or set of items whose specification is optional.
|	Only one of the options separated by a vertical bar can be specified at the same time.
`...`	An ellipsis (...) indicates that the item or items enclosed in ( ) or [ ] immediately preceding the ellipsis may be specified as many times as necessary.

Example font or symbol	Convention
( )	Parentheses indicate the range of items to which the vertical bar (|) or ellipsis (...) is applicable.

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: Version numbers

The version numbers of Hitachi program products are usually written as two sets of two digits each, separated by a hyphen. For example:

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

The version number might be shown on the spine of a manual as *Ver. 2.00,* but the same version number would be written in the program as *02-00*.

Important note on this manual

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

Preface i

Intended readers i
Organization of this manual i
Related publications ii
Conventions: Abbreviations iii
Conventions: Diagrams vi
Conventions: Fonts and symbols vi
Conventions: KB, MB, GB, and TB viii
Conventions: Version numbers viii
Important note on this manual viii

1. Overview 1

1.1 Features of the client 2
1.2 Mechanism of TP1/Client/J operation 5
1.3 Environment of TP1/Client/J 8
1.3.1 Installation 8
1.3.2 Development environment 9
1.3.3 Execution environment 10

2. Functionality 11

2.1 Permanent connection 12
2.1.1 Establishing and releasing a permanent connection 12
2.1.2 Definitions related to the use of a permanent connection 14
2.1.3 Note about using a permanent connection 14
2.2 Remote procedure calls 15
2.2.1 Implementing RPCs 15
2.2.2 Data transfer with an RPC 15
2.2.3 RPC modes 15
2.2.4 Scheduling facility 18
2.2.5 Inter-node load-balancing facility 18
2.2.6 Time monitoring of RPCs 19
2.2.7 RPCs that use the remote API facility 19
2.2.8 RPCs that use the scheduler direct facility 21
2.2.9 RPCs that use the name service 22
2.2.10 RPCs with a communication destination specified 27
2.2.11 Reduction of server workload in the event of synchronous-response RPC timeout 29
2.2.12 Use of ServerSocket for RPCs 30

2.2.13 RPCs using the multi-scheduler facility ... 31
2.3 Transaction control ... 35
2.3.1 Start and synchronization point acquisition of transactions ... 35
2.3.2 Synchronization point acquisition ... 37
2.3.3 Relationship between mode of remote procedure calls and synchronization point ... 41
2.3.4 Acquiring the current transaction identifiers ... 42
2.3.5 Reporting information about the current transaction ... 42
2.3.6 Verifying the transaction synchronization point in the event of an error ... 42
2.3.7 Definitions for TP1/Server ... 44
2.4 TCP/IP communication facility ... 46
2.4.1 Unidirectional sending of messages ... 46
2.4.2 Unidirectional receiving of messages ... 47
2.4.3 Bidirectional sending and receiving of messages ... 50
2.4.4 Received message assembly facility ... 52
2.4.5 Notes about using the TCP/IP communication facility ... 53
2.5 Unidirectional server message reception facility ... 55
2.5.1 Flow of unidirectional message reception facility processing ... 55
2.5.2 Flow of unidirectional message consecutive reception facility processing ... 56
2.5.3 Notes about using the unidirectional message consecutive reception facility ... 57
2.5.4 Releasing the unidirectional message reception wait status ... 57
2.6 TP1/Web connection facility ... 59
2.6.1 Connecting to TP1/Web (starting a session) ... 59
2.6.2 Service requests to TP1/Web ... 59
2.6.3 Releasing a connection to TP1/Web (ending a session) ... 60
2.6.4 RPC facilities for connecting to TP1/Web ... 60
2.7 Dynamic definition changing facility ... 65
2.8 Facility for monitoring TCP/IP connection establishment ... 66
2.8.1 API that uses the connect method internally ... 66
2.8.2 Exceptions that occur when the connect method times out ... 67
2.9 DCCM3 connection facility ... 70
2.9.1 Issuing an ROC to DCCM3 ... 70
2.9.2 TCP/IP communication with DCCM3 ... 75
2.9.3 Sending terminal identification information to a DCCM3 logical terminal. 75
2.10 XA resource service facility ... 80
2.11 Troubleshooting facility ... 82
2.11.1 Contents of trace files ... 82
2.11.2 UAP trace ... 83
2.11.3 Data trace ... 105
2.11.4 Error trace and memory trace ... 105
2.11.5 Method trace ... 114
2.11.6 Debug trace ... 114
2.11.7 Performance analysis trace ... 115

2.12 Data compression facility 128
2.12.1 Effects of the data compression facility 129
2.12.2 Notes about using the data compression facility 129
2.13 Source host specification facility 131
2.14 Receive port fixing facility 133

3. Program Interface 139

3.1 List of APIs 140
3.2 How to use APIs 144
3.2.1 Execution order of APIs 144
3.2.2 Adjustment during TP1/Client/J execution 148
3.2.3 Collecting error information and adjusting facilities 149
3.2.4 Instructing trace output 150

4. Classes Used with TP1/Client/J 153

Class TP1Client 154
Class DCRpcBindTbl 223
Class ErrAcceptCanceledException 224
Class ErrBufferOverflowException 225
Class ErrCollisionMessageException 226
Class ErrClientTimedOutException 227
Class ErrConnfreeException 228
Class ErrConnRefusedException 229
Class ErrFatalException 230
Class ErrHazardException 231
Class ErrHazardNoBeginException 232
Class ErrHeuristicException 233
Class ErrHeuristicNoBeginException 234
Class ErrHostUndefException 235
Class ErrInitializingException 237
Class ErrInvalidArgsException 238
Class ErrInvalidMessageException 239
Class ErrInvalidPortException 240
Class ErrInvalidReplyException 242
Class ErrIOErrException 243
Class ErrMessageTooBigException 244
Class ErrNetDownAtClientException 245
Class ErrNetDownAtServerException 246
Class ErrNetDownException 247
Class ErrNoBeginException 248
Class ErrNoBufsAtServerException 249
Class ErrNoBufsException 250
Class ErrNoSuchServiceException 251
Class ErrNoSuchServiceGroupException 252

Class ErrNotTrnExtendException 253
Class ErrNotUpException 254
Class ErrProtoException 255
Class ErrReplyTooBigException 256
Class ErrRMException 257
Class ErrRollbackException 258
Class ErrRollbackNoBeginException 259
Class ErrSecchkException 260
Class ErrSecurityException 261
Class ErrServerBusyException 262
Class ErrServerTimedOutException 263
Class ErrServiceClosedException 264
Class ErrServiceNotUpException 265
Class ErrServiceTerminatedException 266
Class ErrServiceTerminatingException 267
Class ErrSyserrAtServerException 268
Class ErrSyserrException 269
Class ErrTestmodeException 270
Class ErrTimedOutException 271
Class ErrTMException 273
Class ErrTrnchkException 274
Class ErrTrnchkExtendException 275
Class ErrVersionException 276
Class TP1ClientException 277

5. Definitions 279

5.1 Overview of definitions 280
5.1.1 List of TP1/Client/J environment definitions 280
5.1.2 Rules for definitions 284
5.1.3 Format of path names 286
5.2 Details of TP1/Client/J environment definitions 287
5.2.1 Format 287
5.2.2 Operands 288
5.2.3 Notes about specifying the TP1/Client/J environment definition 315

6. Error Handling 317

6.1 Collecting trace information 318
6.2 Handling network errors 319
6.3 Validity of timer values 320
6.4 Handling other errors 321

Appendix 323

A. Changes During Upgrading 324
A.1 Changes in 07-02 324

A.2 Changes in 07-01 325
A.3 Changes in 07-00 326

Index 329

List of figures

Figure 1-1: Relationships between TP1/Server and TP1/Client/J (for a CUP created by a Java applet) 2
Figure 1-2: Relationship between TP1/Server and TP1/Client/J (for a CUP created by a Java application) 3
Figure 1-3: Relationships between TP1/Server and TP1/Client/J (for a CUP created by a Java servlet) 4
Figure 1-4: Mechanism of TP1/Client operation (1/2) 6
Figure 1-5: Mechanism of TP1/Client operation (2/2) 7
Figure 2-1: Establishing and releasing a permanent connection........ 13
Figure 2-2: Data transfer with an RPC........ 15
Figure 2-3: Processing flow of a synchronous-response RPC 16
Figure 2-4: Processing flow of a non-response type RPC 17
Figure 2-5: Flow of a service request when the remote API facility is used 20
Figure 2-6: Flow of an RPC service request using the scheduler direct facility........ 21
Figure 2-7: Flow of a service request when the name service is used 23
Figure 2-8: Example of issuing an RPC to TP1/Server in the multi-homed host environment26
Figure 2-9: Flow of a service request when an RPC with a communication destination specified is used 28
Figure 2-10: Overview of processing by the server-load reduction facility in the event of a synchronous-response RPC timeout........ 30
Figure 2-11: Relationships between a transaction and an RPC (when Y is specified for the dcrapautoconnect operand)........ 36
Figure 2-12: Chained and unchained modes of transactions 38
Figure 2-13: Rollback of a transaction (when an error occurs in TP1/Server processing)...... 39
Figure 2-14: Rollback of a transaction (when a rollback request method is called)........ 40
Figure 2-15: Relationship between a synchronous-response RPC and the synchronization point........ 41
Figure 2-16: Relationship between a non-response type RPC and the synchronization point 42
Figure 2-17: Method for verifying the transaction synchronization point in the event of an error 44
Figure 2-18: Unidirectional sending of a message........ 47
Figure 2-19: Unidirectional receiving of a message 49
Figure 2-20: Unidirectional receiving of a message when an error occurs........ 50
Figure 2-21: Bidirectional sending and receiving of messages 51
Figure 2-22: Flow of unidirectional message reception facility processing 55
Figure 2-23: Flow of unidirectional message consecutive reception facility processing 56
Figure 2-24: Flow of releasing unidirectional message reception wait status 58
Figure 2-25: Connecting by specifying dcrapautoconnect=Y 61
Figure 2-26: Connecting by specifying dcrapautoconnect=N and defining the dcweburl operand 62

Figure 2-27: Connecting by using an openConnection method in which a URL is specified..63
Figure 2-28: Flow of service execution when using the DC_JGW service set 63
Figure 2-29: Flow of service execution when using the DC_JUSR service set 64
Figure 2-30: Issuing an RPC to a DCCM3 server 73
Figure 2-31: Relationship between CUP and DCCM3 logical terminal (when the terminal identification information setting facility is not used) 76
Figure 2-32: Relationship between CUP and DCCM3 logical terminal (when the terminal identification information setting facility is used) 77
Figure 2-33: Trace collection points in establishing or terminating connection with the RAP-processing listener..119
Figure 2-34: Trace collection points that are used when an API surrogate execution request is issued by the remote API facility 121
Figure 2-35: Trace collection points that are used when issuing an RPC to the schedule server 123
Figure 2-36: Trace collection points that are used when issuing a service information inquiry to the name server...124
Figure 2-37: Overview of database adapter 128
Figure 2-38: When the source host specification facility is not used and when the facility is used...132
Figure 2-39: When the receive port fixing facility is not used (RPC using the scheduler direct facility) 134
Figure 2-40: When the receive port fixing facility is not used (RPC using the name service) 135
Figure 2-41: When the receive port fixing facility is used (RPC using the scheduler direct facility) 136
Figure 2-42: When the receive port fixing facility is used (RPC using the name service) 137

List of tables

Table 1-1: TP1/Client/J files and storage directories 8
Table 1-2: Files used by TP1/Client/J and the required access authorities 9
Table 2-1: Relationship between operand definitions in the TP1/Client/J environment definition and scheduler daemon (RPCs using the scheduler direct facility) 33
Table 2-2: Relationship between operand definitions in the TP1/Client/J environment definition and scheduler daemon (RPCs using the name service) 34
Table 2-3: APIs (methods) that use the connect method internally 66
Table 2-4: Exceptions returned by the API (method) when the connect method times out..... 68
Table 2-5: Communication method supported by TP1/Client/J 80
Table 2-6: Connection modes supported by TP1/Client/J 80
Table 2-7: RPC call modes supported by TP1/Client/J 80
Table 2-8: Options and names for trace files 82
Table 2-9: Information for the openConnection() method and the openConnection(host, port) method 84
Table 2-10: Information for the openConnection(url, flags) method 86
Table 2-11: Information for the rpcCall method 86
Table 2-12: Information for the rpcCallTo method 88
Table 2-13: Information for the closeConnection method 90
Table 2-14: Information for the setDcwatchtim method 90
Table 2-15: Information for the setDccltinquiretime method 91
Table 2-16: Information for the setDccltdelay method 91
Table 2-17: Information for the setDcselint method 91
Table 2-18: Information for the setDccltextend method 91
Table 2-19: Information for the setRpcextend method 92
Table 2-20: Information for the setDchost method 92
Table 2-21: Information for the rpcOpen method 93
Table 2-22: Information for the trnBegin method 93
Table 2-23: Information for the trnInfo method 95
Table 2-24: Information for the getTrnID method 95
Table 2-25: Information for the cltReceive method 96
Table 2-26: Information for the cltSend method 97
Table 2-27: Information for the cltAssemSend method 97
Table 2-28: Information for the cltAssemReceive method 98
Table 2-29: Information for the setConnectInformation method 98
Table 2-30: Information for the acceptNotification method 99
Table 2-31: Information for the cancelNotification method 100
Table 2-32: Information for the openNotification method 100
Table 2-33: Information for the acceptNotificationChained method 101
Table 2-34: Information for the setUapTraceMode method 101
Table 2-35: Information for the setErrorTraceMode method 102

Table 2-36: Information for the setMethodTraceMode method 102
Table 2-37: Information for the setDataTraceMode method 103
Table 2-38: Information for the trnChainedCommit method 104
Table 2-39: Information for the trnUnchainedCommit method 104
Table 2-40: Information for the trnChainedRollback method 104
Table 2-41: Information for the trnUnchainedRollback method 104
Table 2-42: Messages collected in error and memory traces 106
Table 2-43: Correspondence of codes to method names 108
Table 2-44: Correspondence of codes to exception names 112
Table 2-45: Details of the trace collection points when establishing or terminating connection with the RAP-processing listener 117
Table 2-46: Details of the trace collection points when an API surrogate execution request is issued by the remote API facility 119
Table 2-47: Details of the trace collection points when issuing an RPC to the schedule server 121
Table 2-48: Details of the trace collection points during API execution 124
Table 3-1: List of APIs (package and classes) 140
Table 3-2: List of APIs (methods) 140
Table 5-1: List of TP1/Client/J environment definitions 280
Table 5-2: Operation of various methods based on the connection establishment mode 295
Table 5-3: Action of the rpcCall method when 2 or more is specified in the dccltrpcmaxmsgsize operand 309
Table A-1: Additions/deletions to APIs, definitions, and commands in TP1/Client/J 07-02 324
Table A-2: Changes to operation in TP1/Client/J 07-02 324
Table A-3: Additions/deletions to APIs, definitions, and commands in TP1/Client/J 07-01 325
Table A-4: Changes to operation in TP1/Client/J 07-01 325
Table A-5: Additions/deletions to APIs, definitions, and commands in TP1/Client/J 07-00 326
Table A-6: Changes to operation in TP1/Client/J 07-00 326

Chapter

1. Overview

This chapter provides an overview of OpenTP1 client and describes its features.

1.1 Features of the client
1.2 Mechanism of TP1/Client/J operation
1.3 Environment of TP1/Client/J

1.1 Features of the client

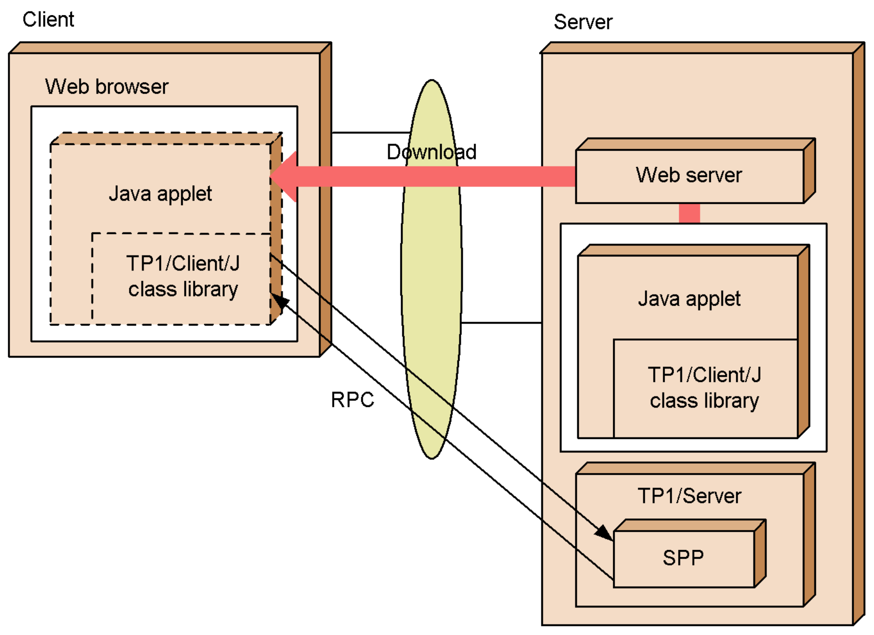
TP1/Client/J makes it possible for Java applets running on a Web browser, Java applications, and Java servlets running on an application server to use remote procedure calls (RPCs) to request services from an OpenTP1 server UAP. A program that is created by a Java applet, Java application, or a Java servlet to request services is called a *CUP* (client user program). A server UAP from which a CUP can request services is called an OpenTP1 *service-providing program* (SPP).

TP1/Client/J enables you to start a transaction from the SPP that was started by a CUP. When you do this, you can apply a distributed OLTP environment to application systems on the Internet or on an intranet. You can also support a variety of platforms with a single CUP because you use Java applets, applications, and servlets to create CUPs. There is no need to create a separate CUP for each platform.

In this manual, TP1/Server Base and TP1/LiNK are referred to collectively as *TP1/ Server*. The TP1/Server Base versions that can process requests from TP1/Client/J are 03-05 and later; earlier versions of TP1/Server Base will not run.

The following figures show the relationships between TP1/Server and TP1/Client/J when a CUP is created by a Java applet, a Java application, or a Java servlet.

Figure 1-1: Relationships between TP1/Server and TP1/Client/J (for a CUP created by a Java applet)



When you create a CUP with a Java applet, you can centralize management of the CUP via the Web server; there is no need to distribute the CUP to clients.

Figure 1-2: Relationship between TP1/Server and TP1/Client/J (for a CUP created by a Java application)

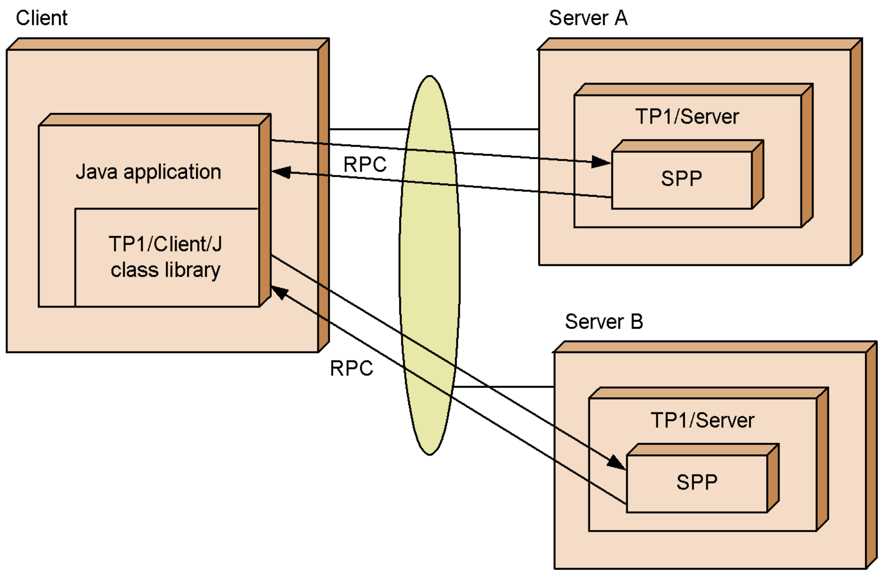
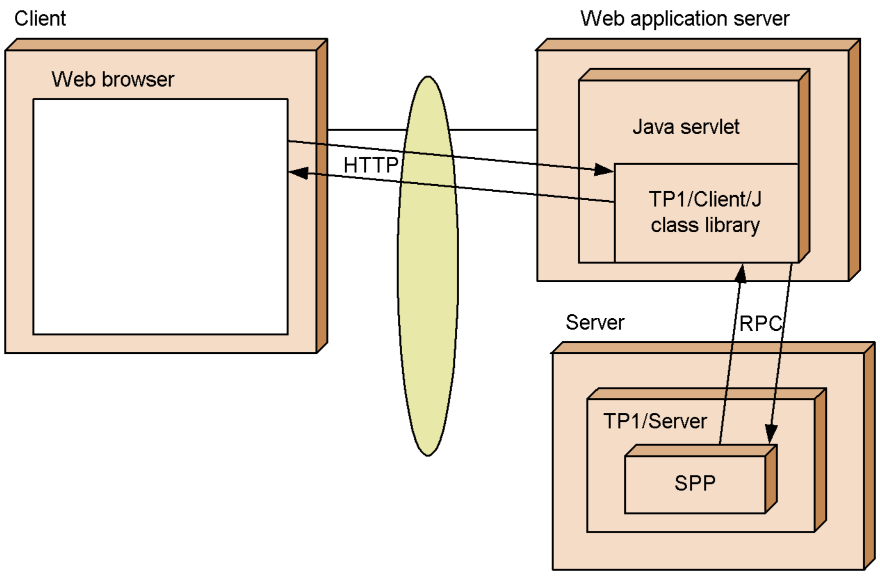


Figure 1-3: Relationships between TP1/Server and TP1/Client/J (for a CUP created by a Java servlet)



1.2 Mechanism of TP1/Client/J operation

When the facilities indicated below execute, TP1/Client/J uses a specific TP1/Server as a gateway. This is called a *remote procedure call* (RPC). For details about RPCs, see the manual *OpenTP1 Programming Guide*. TP1/Client/J supports four types of RPCs:

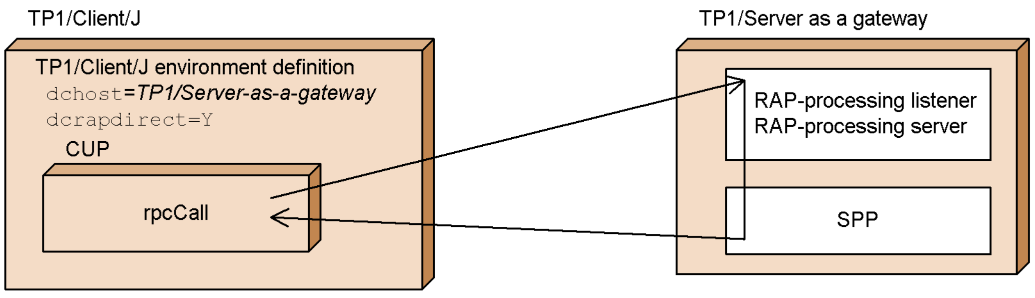
- RPCs that use the remote API facility
- RPCs that use the scheduler direct facility
- RPCs that use the name service
- RPCs with a communication destination specified

You use the `dchost` operand of the TP1/Client/J environment definition to define a TP1/Server as a gateway.

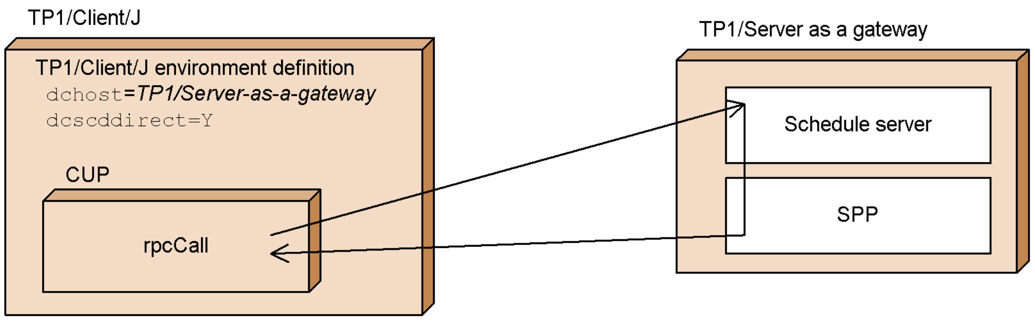
The following figures show the mechanism of TP1/Client/J operation.

Figure 1-4: Mechanism of TP1/Client operation (1/2)

- RPCs that use the remote API facility



- RPCs that use the scheduler direct facility



- RPCs that use the name service

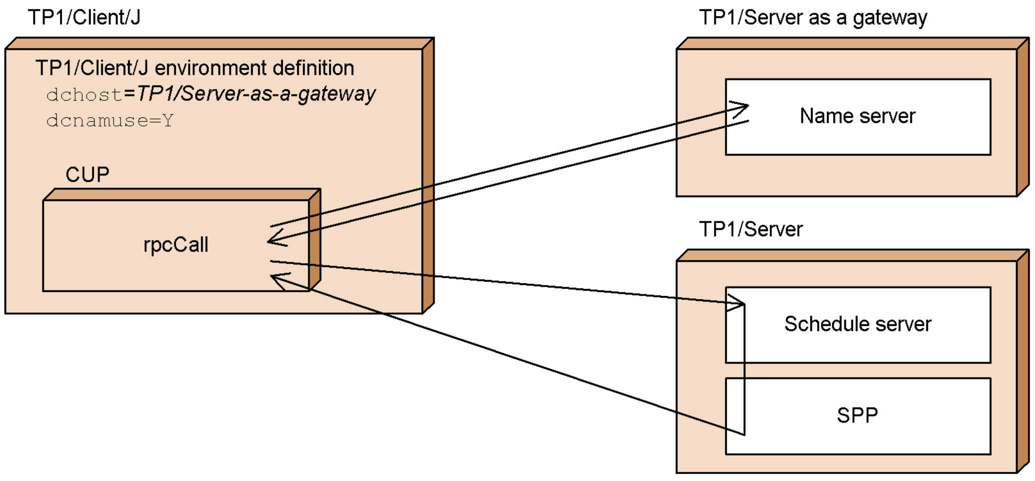
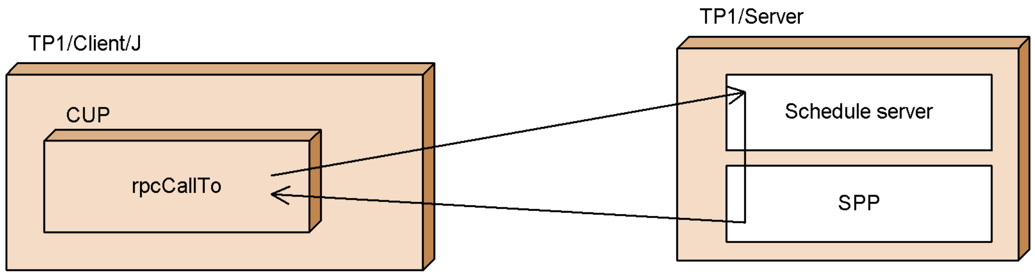


Figure 1-5: Mechanism of TP1/Client operation (2/2)

- RPCs with a communication destination specified



1.3 Environment of TP1/Client/J

TP1/Client/J provides the OpenTP1 server with a class library file for calling applications. This section describes the installation target, development environment, and execution environment of TP1/Client/J.

1.3.1 Installation

(1) Provided medium and format

To support multiple platforms, TP1/Client/J is provided as a Joliet-format CD-ROM. To install the product on a non-PC machine, you must first install it on a PC and then install it on the target machine using a method such as a file transfer in the binary mode.

Because the TP1/Client/J files are stored in long-file-name format, the OS on the target PC must support this file name format.

The TP1/Client/J files are stored in the two directories described below.

Table 1-1: TP1/Client/J files and storage directories

Storage directory	TP1/Client/J file	Description
`LIB`	`TP1Client.jar`	Stores the class libraries of TP1/Client/J in `jar` format.
`SAMPLES`	`AppletSample.html`	Stores sample source codes for applets, applications, and servlets that use TP1/Client/J.
	`AppletSample.java`	
	`ApplicationSample.java`	
	`betran.ini`	
	`ServletSample.java`	

(2) Installation target of the class library file

This section describes the installation target of the class library file.

For developing and executing a Java applet

Save the `TP1Client.jar` file in a desired directory and specify it with the `applet` HTML tag.

For executing a Java application

To use JDK, save the `TP1Client.jar` file in a desired directory and specify the `CLASSPATH` environment variable.

For executing a Java servlet

Save the `TP1Client.jar` file according to the specifications of the Java servlet to be used.

(3) Specifying the file access authority

The following table shows the files that are accessed by TP1/Client/J and the required access authorities.

Table 1-2: Files used by TP1/Client/J and the required access authorities

File type	Access type	Storage location
TP1/Client/J class library	Read	Any location.
TP1/Client/J environment definition	Read	Any location.
UAP trace file	Write	Directory specified in the `dcuaptracepath` operand of the TP1/Client/J environment definition or in the `path` argument of the `setUapTraceMode` method.
Data trace file	Write	Directory specified in the `dcdatatracepath` operand of the TP1/Client/J environment definition or in the `path` argument of the `setDataTraceMode` method.
Method trace file	Write	Directory specified in the `dcmethodtracepath` operand of the TP1/Client/J environment definition or in the `path` argument of the `setMethodTraceMode` method.
Error trace file	Write	Directory specified in the `dcerrtracepath` operand of the TP1/Client/J environment definition or in the `path` argument of the `setErrorTraceMode` method.
Debug trace file	Write	`TP1clientJ` directory under the home directory of the user executing the Java VM.

To apply the security manager to the Java VM on which a client program using TP1/Client/J is running, specify appropriate permissions for the security policy file so that the TP1/Client/J class library can access these files. In the case of Java servlets and EJB, some application server products with these containers may be using the security manager by default for security purposes. Check the specifications of the application server you are using and specify appropriate permissions.

1.3.2 Development environment

To run TP1/Client/J, your environment must support development of Java applets, applications, and servlets (such as by providing a Java compiler).

JDK version

Java$^{(TM)}$2 Software Development Kit, Standard Edition, Version 5.0 (JDK 5.0) or later is supported.

1.3.3 Execution environment

TP1/Server server version

TP1/Server Base version 03-05 or later can process requests from TP1/Client/J.

Cosminexus Application Server

To use TP1/Client/J from a Java servlet, EJB, or JSP in the J2EE server mode# of Cosminexus Application Server, you must have specified the following settings in advance:

1. Edit the option definition file of the J2EE server# and specify the following character string:

   `add.class.path=<`*TP1/Client/J-installation-directory*`>/TP1Client.jar`

2. Edit the option definition file of the server management command# and specify the following character string:

   In Windows

   `set USRCONF_JVM_CLASSPATH=<`*TP1/Client/J-installation-directory*`>/TP1Client.jar`

   In UNIX

   `set USRCONF_JVM_CLPATH=<`*TP1/Client/J-installation-directory*`>/TP1Client.jar`

   Specify the path name of the installation directory in the format supported by the operating system being used.

#

For details about the J2EE server mode, option definition of the J2EE server, and option definition of the server management commands, see the manuals *Cosminexus Function Description* and *Cosminexus Reference - Definition*.

Chapter

2. Functionality

This chapter describes the OpenTP1 client.

2.1 Permanent connection
2.2 Remote procedure calls
2.3 Transaction control
2.4 TCP/IP communication facility
2.5 Unidirectional server message reception facility
2.6 TP1/Web connection facility
2.7 Dynamic definition changing facility
2.8 Facility for monitoring TCP/IP connection establishment
2.9 DCCM3 connection facility
2.10 XA resource service facility
2.11 Troubleshooting facility
2.12 Data compression facility
2.13 Source host specification facility
2.14 Receive port fixing facility

2.1 Permanent connection

TP1/Client/J enables you to send and receive messages while a connection is established between a CUP (Java applet, application, or servlet using TP1/Client/J) and a RAP-processing server. Such a connection is called a *permanent connection*. When you execute RPCs that use the remote API facility, chained RPCs, or the transaction facility, you much ensure that a permanent connection has been established.

By establishing a permanent connection, you minimize the number of control packets for establishing and releasing connection, thereby improving communications efficiency.

2.1.1 Establishing and releasing a permanent connection

You call the `openConnection` method to establish a permanent connection. You specify the location of the target RAP-processing server in an argument of the `openConnection` method or in the `dchost` operand in the TP1/Client/J environment definition and the `dcrapport` operand. If there is a firewall between the CUP and the RAP-processing listener and server, you must specify the location of the firewall instead of the RAP-processing server.

If the `openConnection` method returns an exception, the permanent connection has not been established.

TP1/Client/J can establish only one permanent connection per CUP at a time. TP1/Client/J supports two permanent connection modes, depending on the management method:

Non-auto connect mode

In the non-auto connect mode, TP1/Client/J calls explicitly the `openConnection` method with a CUP and establishes a connection with the RAP-processing server or RAP-processing listener of TP1/Server. If the `rpcCall` method is called before the `openConnection` method is called, the `rpcCall` method returns `ErrProtoException`. To release the connection, call the `closeConnection` method with CUP.

Auto connect mode

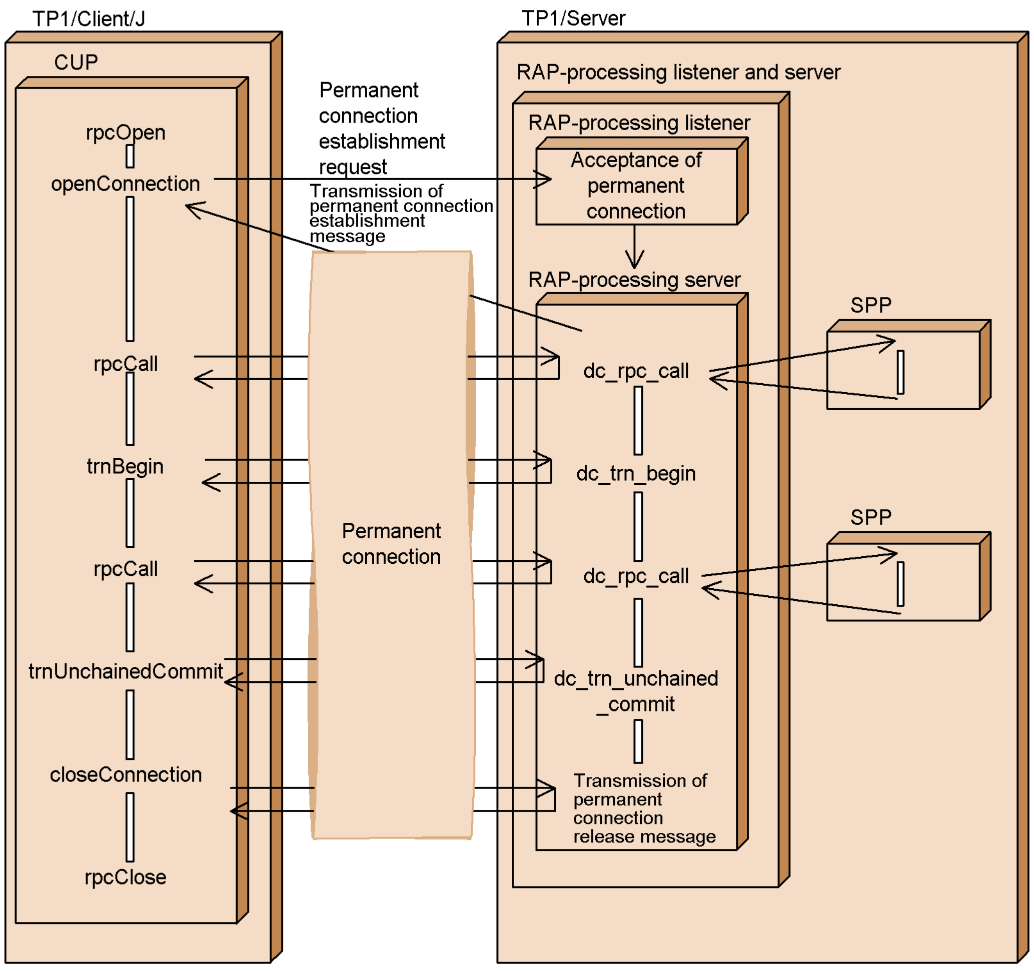
The auto connect mode manages the connection with TP1/Client/J. If TP1/Client/J determines that a connection has not been established when the first `rpcCall` or `trnBegin` method is called, it automatically establishes a permanent connection with the RAP-processing listener and server. In this case, there is no need to call the `openConnection` method. If the `openConnection` method is called, the method returns `ErrProtoException`. You can release the connection within the `rpcClose` method. To forcibly release a TP1/Client/J-managed connection, call the `closeConnection` method.

You use the `dcrapautoconnect` operand in the TP1/Client/J environment definition or the `setRpcextend` method to specify the mode that is to be used (non-auto connect mode or auto connect mode).

TP1/Client/J uses a permanent connection for message transmission until the permanent connection is released by the `closeConnection` method. However, some errors will also cause the permanent connection to be released.

The following figure shows the procedure for establishing and releasing a permanent connection.

Figure 2-1: Establishing and releasing a permanent connection



2.1.2 Definitions related to the use of a permanent connection

To use a permanent connection, you must specify the following definitions as necessary:

- TP1/Client/J environment definition
    - `dccltinquiretime`
    - `dchost`
    - `dcrapautoconnect`
    - `dcrapdirect`
    - `dcrapport`
- RAP-processing listener service definition

  For details about the RAP-processing listener service definition, see the manual *OpenTP1 System Definition*.

2.1.3 Note about using a permanent connection

You cannot establish a permanent connection from within a transaction. You can generate transactions only after you have established a permanent connection.

2.2 Remote procedure calls

This section describes remote procedure calls (RPCs) from a CUP to an SPP.

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

2.2.1 Implementing RPCs

You use the user service structure definition of TP1/Server or the `dcsvstart` command to start an SPP.

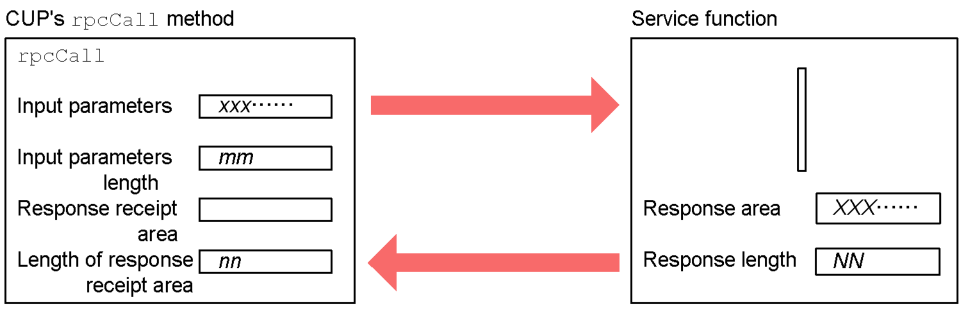
The procedure is to call a method that requests service from the CUP and issue a request to the SPP. To request service, you use the `rpcCall` method, which specifies in its parameters the service group name and service name of the SPP.

2.2.2 Data transfer with an RPC

To transfer data with an RPC, you call the `rpcCall` method specifying the SPP service group name, service name, input parameters, input parameters length, service response storage area, and response length. The maximum response length that can be specified with the `rpcCall` method is 1 MB. You can change this value using the `dccltrpcmaxmsgsize` operand of the TP1/Client/J environment definition.

The following figure shows data transfer with an RPC.

Figure 2-2: Data transfer with an RPC



Note: You must ensure that the length of the response from the service function does not exceed the value set in the `rpcCall` method (value of *nn*).

2.2.3 RPC modes

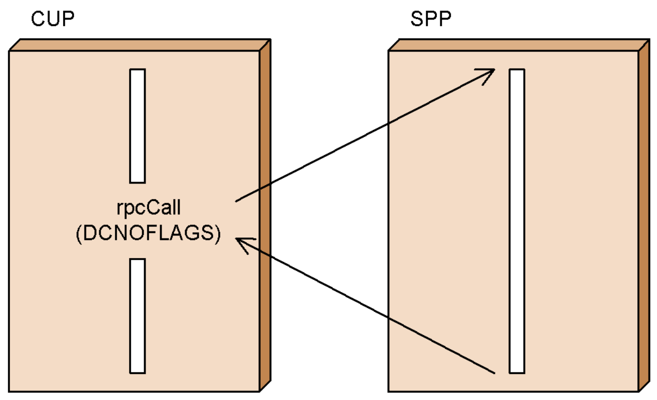
TP1/Client/J supports three RPC modes: synchronous-response RPC, non-response type RPC, and chained RPC.

(1) Synchronous-response RPC

In this mode, a CUP sends an inquiry message to an SPP and receives a response message. The CUP waits for the processing results to be returned from the SPP before executing the next processing. If the same service is called more than once, the server process is scheduled each time.

The following figure shows the processing flow of a synchronous-response RPC.

Figure 2-3: Processing flow of a synchronous-response RPC



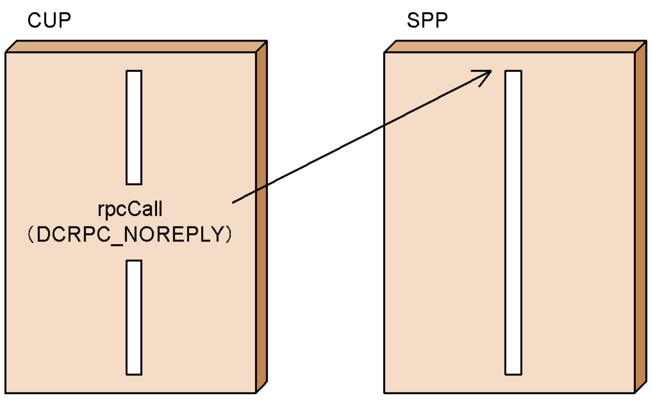
(2) Non-response type RPC

In this mode, a CUP sends an inquiry message to an SPP but does not receive a response message. Once it has sent an inquiry message to the SPP, the CUP immediately executes the next processing without waiting for the processing results from the SPP.

In the case of a non-response type RPC, you cannot receive an error in the event of a communication error or service error.

The following figure shows the processing flow of a non-response type RPC.

Figure 2-4: Processing flow of a non-response type RPC



(3) Chained RPC

In this mode, a CUP that uses the remote API facility sends an inquiry message to an SPP and receives a response message. The CUP waits for the processing results to be returned from the SPP before executing the next processing. The difference from a synchronous-response RPC is that a chained RPC can fix the server process. This means that when multiple inquiry messages are sent to the same service by means of chained RPCs, the server process is not rescheduled.

Chained RPCs require fewer user processes per transaction, thereby reducing the workload of transaction processing. A chained RPC used as a transaction functions in a single global transaction.

Starting chained RPCs

To issue a service request for chained RPCs, specify `DCRPC_CHAINED` in the `flags` parameter of the `rpcCall` method that requests the service. When a service is requested with this value specified, the SPP identifies it for chained RPCs and acquires a process. For the second and subsequent service requests, you must also specify `DCRPC_CHAINED` in the `flags` argument.

Terminating chained RPCs

The following methods are provided for terminating chained RPCs:

- Execute the `rpcCall` method (synchronous-response RPC) with `DCNOFLAGS` specified in the `flags` parameter for the service group that has been executing the chained RPCs.
- Complete the global transaction that has been executing the chained RPCs by synchronization point processing (commit or rollback).

Note

If you call the `closeConnection` or `rpcClose` method without executing a synchronous-response RPC, the following occurs:

- Outside the global transaction range

  The process executing the service is locked until the chained RPC timeout occurs.

- Within the global transaction range

  Implicit commit occurs and the chained RPC is terminated.

Time monitoring of chained RPCs

A UAP whose service is requested with a chained RPC monitors the time until the next service request arrives or until transaction synchronization point processing occurs after a response is returned to the CUP. If no service request or synchronization point processing request is received within this monitoring interval, the UAP determines that an error has occurred in the CUP and terminates the SPP abnormally. You specify the monitoring interval in the `watch_next_chain_time` operand of the user service definition.

2.2.4 Scheduling facility

The scheduling facility of TP1/Server is also applicable to service requests from a CUP to an SPP. TP1/Server creates a schedule queue for each service group of an SPP and schedules the service requests.

2.2.5 Inter-node load-balancing facility

OpenTP1 provides a facility for distributing the workload among nodes to prevent RPC requests from becoming concentrated on a specific node. This is called the *inter-node load-balancing facility*.

To use the inter-node load-balancing facility, the following conditions must be satisfied:

- A user server is active that provides the same service to multiple nodes.
- Each OpenTP1 node defines the other nodes in the `all_node` operand in the system common definition and shares information about the user server (name information) that is active in each OpenTP1 node.

This section describes the definitions, processing, and RPC processing related to the use of the inter-node load-balancing facility of OpenTP1 for TP1/Client/J and TP1/Server. Note that the TP1/Client/J facility balances workloads based only on TP1/Server-side determination.

(1) Load balancing based on a TP1/Server determination

The schedule service of TP1/Server distributes the workload to appropriate nodes on the basis of the status of node schedules.

(a) Definitions for TP1/Client/J

Specify `dcscddirect=Y` in the TP1/Client/J environment definition.

This enables TP1/Client/J to issue a load-balancing request to the schedule service of TP1/Server. In the definitions for TP1/Client/J, specify the OpenTP1 node whose schedule service is to make the determination.

If there are multiple OpenTP1 nodes that can issue a scheduling request, scheduling is requested in the order specified in the `dchost` operand. To randomly select a TP1/Server that is to issue a scheduling request, instead of following the order specified in the `dchost` operand, specify `dchostselect=Y` in the TP1/Client/J environment definition.

(b) Definitions for TP1/Server

In the definitions for TP1/Server, do one of the following:

- Specify the following operands in the schedule service definition:

  `scd_this_node_first = N` (default)

  `scd_announce_server_status = Y` (default)

- Omit the schedule service definition.

2.2.6 Time monitoring of RPCs

When you issue a synchronous-response RPC, you can monitor the length of time until a response message is received.

You use the `dcwatchtim` operand in the TP1/Client/J environment definition to specify a monitoring interval.

You can also specify the monitoring interval by calling the `setDcwatchtim` method from the CUP. To change the monitoring interval as appropriate for the service to be requested, you must set a new monitoring interval before issuing the RPC. If no response message is received within the specified monitoring interval, the RPC returns an error.

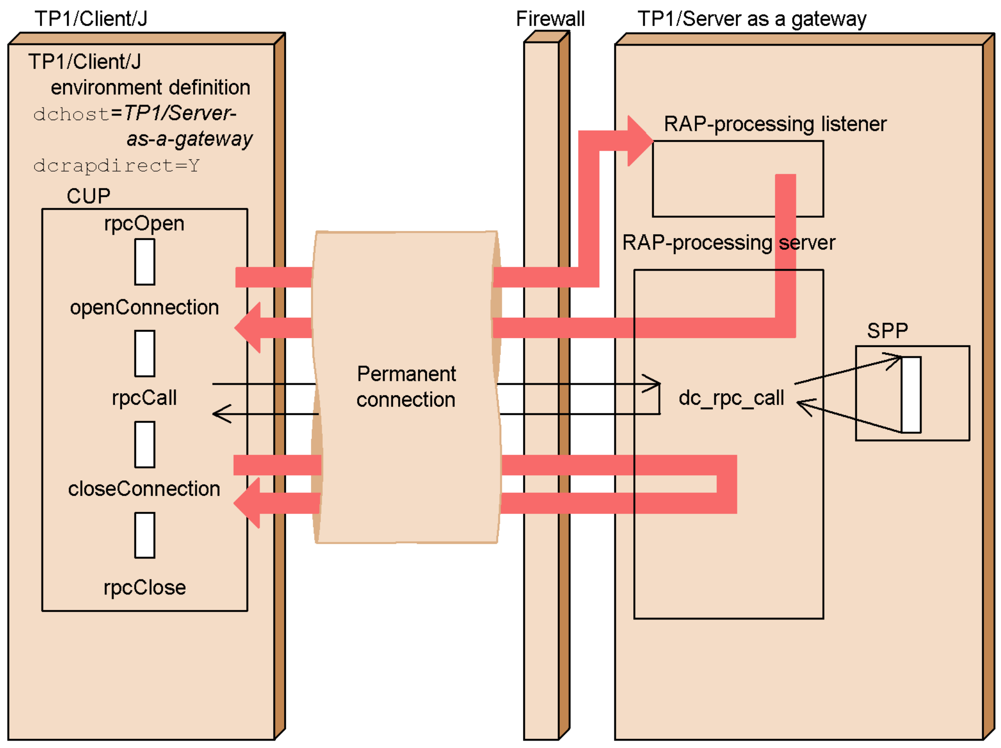
2.2.7 RPCs that use the remote API facility

TP1/Client/J enables you to establish a permanent connection between a CUP and the TP1/Server, transfer an API issued by the CUP to the TP1/Server, and have it executed by a TP1/Server process. This functionality is called the *remote API facility*. When you use the remote API facility, you can also issue service requests to UAPs inside the firewall. In the case of a Java applet, because of Java security limitations, connection can be established only with a RAP-processing server that is active in the TP1/Server on the Web server to which the Java applet belongs. If an attempt is made to establish connection with a RAP-processing server that is active outside the TP1/Server on the Web server to which the Java applet belongs, the security manager returns

`SecurityException`. An RPC that uses the remote API facility can use DCCM3 and TMS-4V/SP, as well as TP1/Server.

The following figure shows the flow of a service request when the remote API facility is used.

Figure 2-5: Flow of a service request when the remote API facility is used



1. Create an instance of the `TP1Client` class that is provided by TP1/Client/J.
2. Call the `openConnection` method to establish connection with the TP1/Server's RAP-processing server.
3. Call the `rpcCall` method to issue a service request to the specified SPP via the RAP-processing server.

   You can call the `rpcCall` method as many times as necessary while the connection is established.
4. Call the `closeConnection` method to release the connection with the TP1/Server's RAP-processing server.

Note

If you issue a service request to a server that is managed by the security facility of DCE using the RAP-processing server, `ErrSecchkException` is returned.

When using the remote API facility with the `dccltrpcmaxmsgsize` operand specified in the TP1/Client/J environment definition, if you issue a service request to the SPP of a TP1/Server for which the `rpc_max_message_size` operand is not specified in the system common definition, an RPC error is returned.

2.2.8 RPCs that use the scheduler direct facility

An RPC that uses the scheduler direct facility constitutes a service request that can be executed from a Java application or servlet. It cannot be executed from a Java applet due to security limitations.

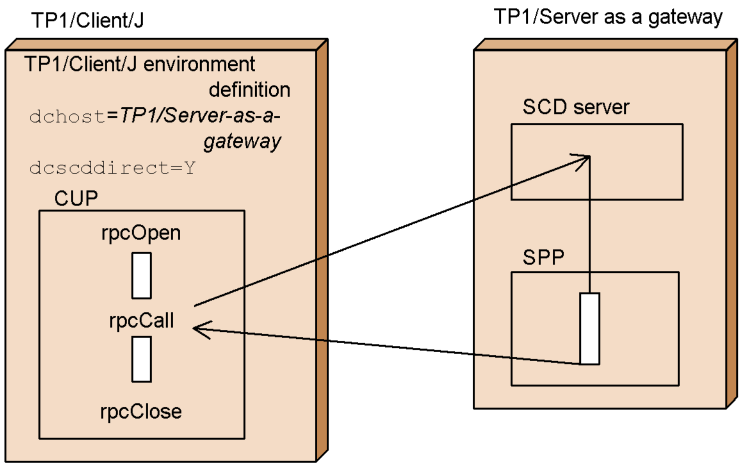
This section explains the RPC flow that uses the scheduler direct facility and the definition necessary for distributing the service request destination schedulers among RPCs.

(1) RPC flow using the scheduler direct facility

To issue an RPC that uses the scheduler direct facility, specify `dcscddirect=Y` in the TP1/Client/J environment definition. Also specify the `dchost` and `dcscdport` operands in the JP1/Software Distribution Client environment definition or specify the SCD server (schedule server) using the `setDchost` method. Before calling an `rpcCall` method, you must call an `rpcOpen` method. At the end of the CUP, you must call an `rpcClose` method.

The following figure shows the flow of an RPC service request using the scheduler direct facility.

Figure 2-6: Flow of an RPC service request using the scheduler direct facility



1. Create an instance of the `TP1Client` class that is provided by TP1/Client/J.

2. Call the `rpcOpen` method to initialize the RPC environment of the CUP.
3. Call the `rpcCall` method to issue a service request to the specified SPP via the TP1/Server's SCD server.

   You can call the `rpcCall` method as many times as necessary while the connection is established (until an `rpcClose` method is called after the `rpcOpen` method was called).
4. Call the `rpcClose` method to release the RPC environment.

When you use this facility, you cannot issue an RPC to an SPP that receives requests from a socket. Furthermore, if you use this facility with the `dccltrpcmaxmsgsize` operand specified in the TP1/Client/J environment definition, an error may occur at the communication destination TP1/Server node.

(2) Definition for distributing the service request destination schedulers among RPCs

When RPCs use the scheduler direct facility, you can distribute the service request destination schedulers among RPCs by switching the RPC in the round-robin mode. To distribute the service request destination schedulers among RPCs, specify `dcscdhostchange=Y` in the TP1/Client/J environment definition.

2.2.9 RPCs that use the name service

An RPC that uses the name service constitutes a service request that can be executed from a Java application or servlet. It cannot be executed from a Java applet due to security limitations.

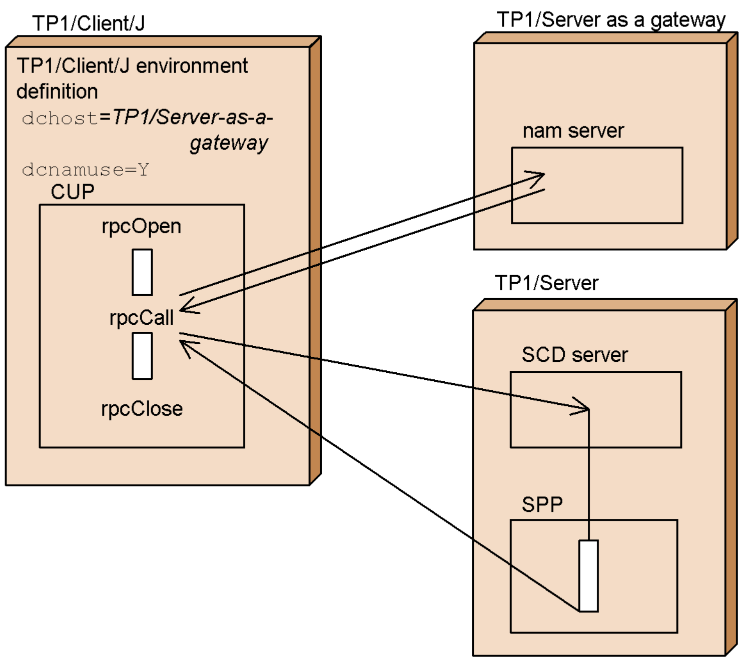
This section explains the RPC flow that uses the name service and the definition necessary for distributing the service request destination schedulers among RPCs.

(1) RPC flow using the name service

To issue an RPC that uses the name service, specify `dcnamuse=Y` in the TP1/Client/J environment definition. Also specify the `dchost` and `dcnamport` operands in the TP1/Client/J environment definition or specify the nam server with the `setDchost` method. Before calling an `rpcCall` method, you must call an `rpcOpen` method. At the end of the CUP, you must call an `rpcClose` method.

The following figure shows the flow of a service request when the name service is used.

Figure 2-7: Flow of a service request when the name service is used



1. Create an instance of the `TP1Client` class that is provided by TP1/Client/J.
2. Call the `rpcOpen` method to initialize the RPC environment of the CUP.
3. Call the `rpcCall` method to issue a service request to the specified SPP.#
4. Call the `rpcClose` method to release the RPC environment.

#

The internal processing of the `rpcCall` method is as follows:

1. Establishes connection with the nam server that is defined with the `dchost` and `dcnamport` operands.
2. Sends a request to acquire service information and release connection with the nam server.
3. After receiving a connection establishment request from the nam server to send a response message, establishes connection and receives the service information.
4. Releases connection with the nam server and uses the service information in the response message from the nam server to establish connection with the SCD server on the TP1/Server that is executing the service.

5. After establishing connection with the SCD server, sends the service request and releases the connection.
6. After receiving a connection establishment request from an SPP to send a response message, establishes the connection and receives the response message.
7. Releases the connection with the SPP.

When you use this facility, you cannot issue an RPC to an SPP that receives requests from a socket. If you specify a value of `2` or greater for the `dccltrpcmaxmsgsize` operand of the TP1/Client/J environment definition, an RPC using the name service collects information only from services that are running on a TP1/Server that supports the `rpc_max_message_size` operand of the system common definition. Even when the input parameter length for the service exceeds the value in the `rpc_max_message_size` operand of the system common definition at the service request destination, the service request is executed. You cannot issue an RPC to an SPP on a TP1/Server that does not support the `rpc_max_message_size` operand of the system common definition.

(2) Definition for distributing the service request destination schedulers among RPCs

When RPCs use the name service, you can store the information on the service request destination schedulers from the name server into the cache, and distribute the service request destination schedulers among RPCs by referencing the information in the cache. To distribute the service request destination schedulers among RPCs, the following definitions are necessary for TP1/Client/J and TP1/Server.

(a) Definition for TP1/Client/J

To distribute the service request destination schedulers among RPCs, specify `dccltloadbalance=Y` in the TP1/Client/J environment definition.

With this setting, TP1/Client/J acquires information on multiple service request destination schedulers from the name server and stores the information on the service request destination scheduler having the lowest load level in the cache. One or more service request destination schedulers may be stored in the cache.

If multiple service request destination schedulers are stored in the cache, the first service request destination scheduler is selected at random. If the service request from an RPC is the second or subsequent one, the cache is referenced to obtain information on service request destination schedulers, and service request destination schedulers are switched in the round-robin mode and distributed among the RPCs.

Specifying a time limit for the cache

Using the `dccltcachetim` operand of the TP1/Client/J environment definition, you can specify a time limit for retaining the information on the service request destination schedulers in the cache.

After storing information on the service request destination schedulers in the cache, if the cache expiration date is reached, the information on the service request destination schedulers in the cache is discarded. The cache is updated when information on the service request destination schedulers is subsequently acquired and stored in the cache.

The `dccltcachetim` operand of the TP1/Client/J environment definition is valid only when `dccltloadbalance=Y` is specified in the TP1/Client/J environment definition.

(b) Definition for TP1/Server

When operands are specified in the TP1/Server definitions listed below, the scheduler load information is also acquired when information on the service request destination schedulers is acquired from the name server.

Schedule service definition

- `scd_announce_server_status=Y` (default)#1

User service definition or user service default definition

- `loadcheck_interval`
- `levelup_queue_count`#2
- `leveldown_queue_count`#2

#1

If `N` is specified in this operand, the load level of the scheduler is always set to `LEVEL0` (low load level). If a scheduler whose load level is always `LEVEL0` becomes a service request destination scheduler of TP1/Client/J, it is always selected, regardless of its actual load condition.

#2

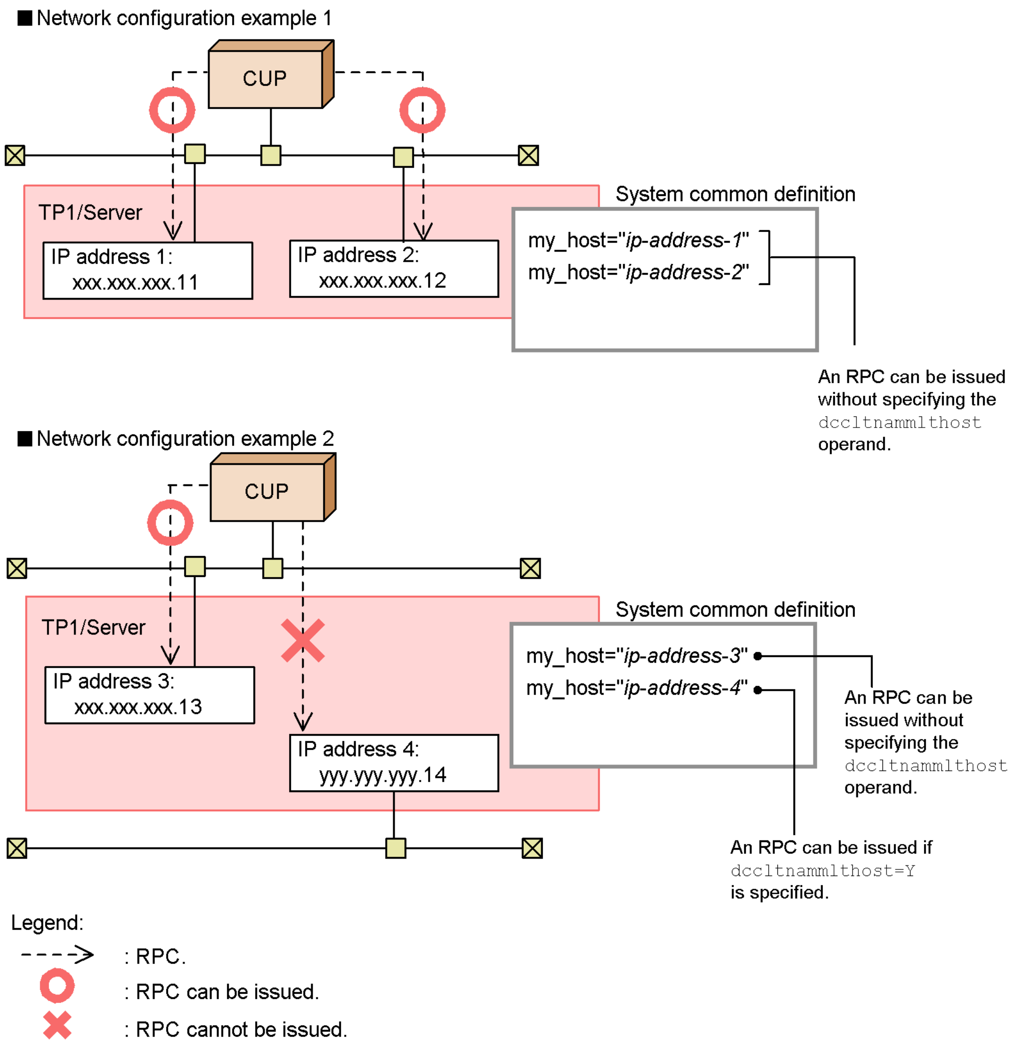
Specify this operand if the `Specify` scheduler's load level is to be determined by the number of remaining service requests.

(3) Definition for issuing an RPC to TP1/Server in the multi-homed host environment

If the communication destination TP1/Server is in the multi-homed host environment, an RPC that uses the name service may encounter a communication error. This is caused when the network where the CUP resides cannot communicate with the network adapter that corresponds to the host name (IP address) specified by the `my_host` operand in the system common definition of the communication destination TP1/Server.

To avoid this problem, specify `dccltnammlthost=Y` in the TP1/Client/J environment definition. The following figure shows two network configuration examples and shows whether the `dccltnammlthost` operand of TP1/Client/J environment definition is required in each of these network configurations.

Figure 2-8: Example of issuing an RPC to TP1/Server in the multi-homed host environment



■Network configuration example 1
CUP
TP1/Server
IP address 1: xxx.xxx.xxx.11
IP address 2: xxx.xxx.xxx.12
System common definition
my_host="ip-address-1"
my_host="ip-address-2"
An RPC can be issued without specifying the dccltnammlthost operand.
■Network configuration example 2
CUP
TP1/Server
IP address 3: xxx.xxx.xxx.13
IP address 4: yyy.yyy.yyy.14
System common definition
my_host="ip-address-3"
my_host="ip-address-4"
An RPC can be issued without specifying the dccltnammlthost operand.
An RPC can be issued if dccltnammlthost=Y is specified.
Legend:
: RPC.
: RPC can be issued.
: RPC cannot be issued.

For the CUP in network configuration example 2 to be able to communicate with IP address 4, `dccltnammlthost=Y` must be specified in the TP1/Client/J environment definition. Without this specification, a communication error occurs between the CUP and IP address 4.

If the communication destination TP1/Server is in the multi-homed host environment, and the service request destination SPP resides on the same TP1/Server, specifying `dccltnammlthost=Y` in the TP1/Client/J environment definition enables the issuance of an RPC. If a TP1/Server residing on a network to which no CUP is connected is specified in the `all_node` operand of the system common definition of the communication destination TP1/Server, an attempt to issue an RPC to an SPP on the first TP1/Server causes an error. If the communication destination TP1/Server is not in the multi-homed host environment, you can issue an RPC regardless of the specification in the `dccltnammlthost` operand of the TP1/Client/J environment definition.

2.2.10 RPCs with a communication destination specified

An RPC with a communication destination specified enables you to specify explicitly the server that is to execute the service request (target OpenTP1 node).

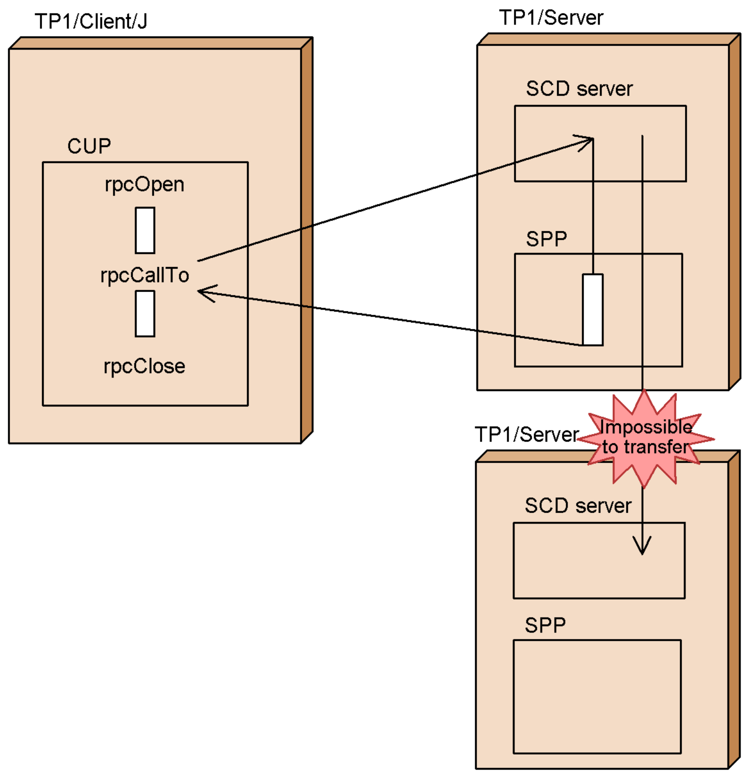
When an RPC with a communication destination specified is used, a service request is executed at the target OpenTP1 node specified as the server that executes the service request. The service request will not be transferred to any other OpenTP1 node even if the specified OpenTP1 node is inactive or shut down. You cannot use the inter-node load-balancing facility with this type of RPC.

To use an RPC with a communication destination specified, call the `rpcCallTo` method. The `rpcCallTo` method executes the service request directly on the TP1/Server schedule service under the host name that is contained in the `DCRpcBindTbl` object.

Note that you cannot use this facility together with an RPC that uses the remote API facility.

The following figure shows the flow of a service request when an RPC with a communication destination specified is used.

Figure 2-9: Flow of a service request when an RPC with a communication destination specified is used



1. Create an instance of the `TP1Client` class that is provided by TP1/Client/J.
2. Call the `rpcOpen` method to initialize the RPC environment of the CUP.
3. Call the `rpcCallTo` method and issue a service request to the specified SPP via the TP1/Server's SCD server.

   The service request is processed by the SPP in the same node, regardless of the status of the SPP at the communication destination.

   You can call the `rpcCallTo` method as many times as necessary until the `rpcClose` method is called after the `rpcOpen` method.
4. Call the `rpcClose` method to release the RPC environment.

When you use this facility, you cannot issue an RPC to an SPP that receives requests from a socket. Furthermore, when you use this facility with the `dccltrpcmaxmsgsize` operand specified in the TP1/Client/J environment definition, an error may occur at the communication destination TP1/Server node.

2.2.11 Reduction of server workload in the event of synchronous-response RPC timeout

When an `rpcCall` method is called from a TP1/Client/J CUP, the TP1/Server accepts a service request.

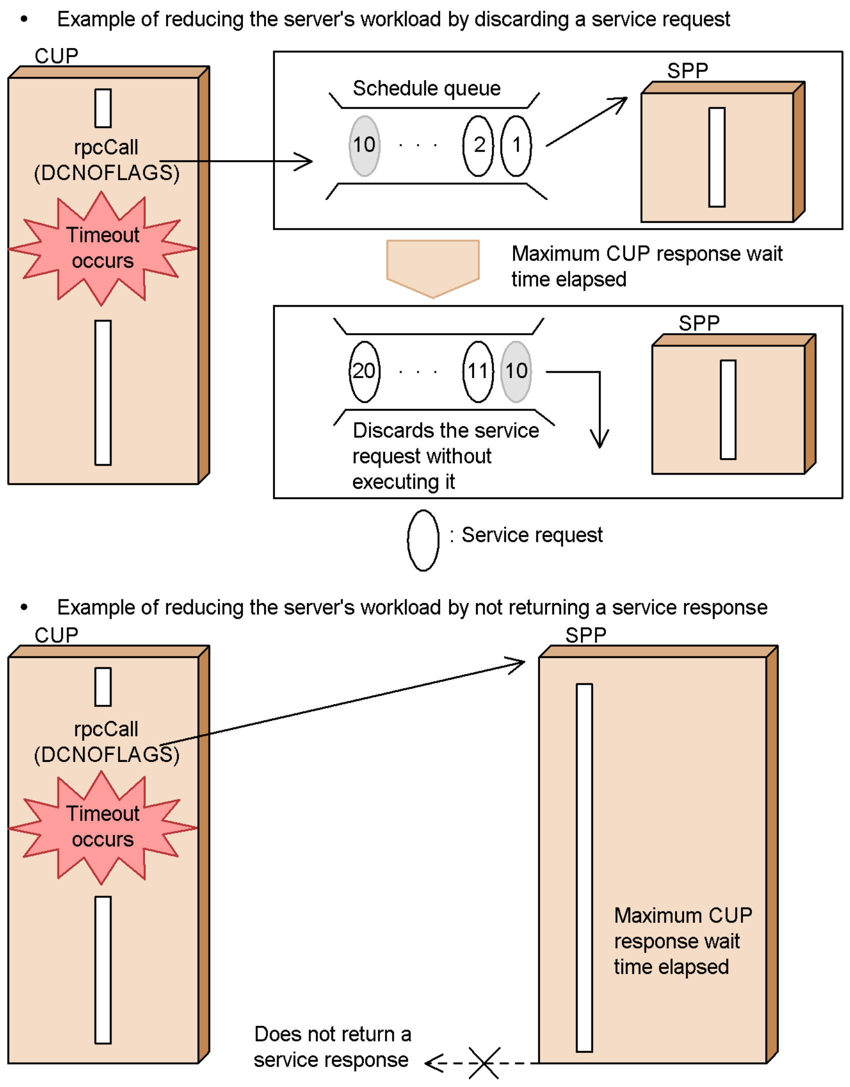
Because this request may be delayed for some reason, such as SPP execution wait time, execution time, or communication error, TP1/Client/J monitors for errors by setting a maximum response wait time.

On the other hand, TP1/Server does not recognize the maximum response time of TP1/Client/J. Even if TP1/Client/J detects a timeout, TP1/Server may be continuing with service processing.

By using the server-load reduction facility in the event of a synchronous-response RPC timeout, you can reduce unneeded TP1/Server processing as mentioned above. You use the `dcwatchtimrpcinherit` operand in the TP1/Client/J environment definition to specify whether or not the server-load reduction facility is to be used in the event of a synchronous-response RPC timeout.

The following figure provides an overview of processing by the server-load reduction facility in the event of a synchronous-response RPC timeout.

Figure 2-10: Overview of processing by the server-load reduction facility in the event of a synchronous-response RPC timeout


• Example of reducing the server's workload by discarding a service request
CUP
rpcCall (DCNOFLAGS)
Timeout occurs
Schedule queue
10
2
1
SPP
Maximum CUP response wait time elapsed
SPP
20
11
10
Discards the service request without executing it
: Service request
• Example of reducing the server's workload by not returning a service response
CUP
SPP
rpcCall (DCNOFLAGS)
Timeout occurs
Maximum CUP response wait time elapsed
Does not return a service response

2.2.12 Use of ServerSocket for RPCs

`ServerSocket` is used to issue an RPC that uses the scheduler direct facility or the name service or an RPC with a communication destination specified in conjunction with TP1/Client/J. To apply the security manager to the Java VM on which a client program using TP1/Client/J is running, specify appropriate permissions to the security policy file so that the class library of TP1/Client/J can use the `ServerSocket` functions (`connect`, `listen`, and `accept`). In the case of Java servlets and EJB, some application server products with these containers may be using the security manager by default for security purposes. Check the specifications of the application server you are using and specify appropriate permissions.

2.2.13 RPCs using the multi-scheduler facility

When a service request is issued from a CUP to an SPP that uses a schedule queue (queue-receiving server), the scheduler daemon at the node where the target SPP is located receives the service request message and stores it in the corresponding SPP's schedule queue. A scheduler daemon is a system daemon that provides schedule service.

A large service request message is divided into multiple segments of a specific length and is then sent to the scheduler daemon. The scheduler daemon assembles the service request message and stores it in the queue-receiving server's schedule queue. Because there can be only one scheduler daemon process per OpenTP1 system, the scheduler daemon cannot receive another service request message until it completes reception processing on the current segmented service request message. If a large service request message is sent over a low-speed line, scheduling of other service requests may be delayed. As the system size increases and machine and network performance improve, efficient scheduling may become a problem. If this occurs, you can prevent scheduling delays by starting multiple service request reception-dedicated daemon processes separately from the conventional scheduler daemon, and by performing multiple service request message reception processes concurrently. This is called the *multi-scheduler facility*. Hereafter, the conventional scheduler daemon is called the *master scheduler daemon*, and the service request reception-dedicated daemon is called the *multi-scheduler daemon*.

For details about the system configuration that must be evaluated in order to use the multi-scheduler facility, see the manual *OpenTP1 Version 7 Programming Guide*.

(1) How to select a multi-scheduler daemon at random

Using the multi-scheduler facility enables random selection of an available multi-scheduler daemon from the multiple active multi-scheduler daemons that have been provided; a service request can then be sent. You can select a multi-scheduler daemon at random and execute the scheduler direct facility or an RPC using the name service.

(a) RPCs using the scheduler direct facility

This subsection describes execution of RPCs using the scheduler direct facility when `y` is specified in the `dcscddirect` operand in the TP1/Client/J environment definition.

TP1/Client/J enables you to randomly select a multi-scheduler daemon without having to send a query to the name service of TP1/Server as a gateway. This reduces the amount of communication and the workload of the name service.

To randomly select a multi-scheduler daemon, specify the following port number in the `dchost` or `dcscdport` operand in the TP1/Client/J environment definition:

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

You must also specify in the `dcscdmulticount` operand in the TP1/Client/J environment definition the number of multi-scheduler daemon processes that can be started by TP1/Server. The port number of the multi-scheduler daemon that is to send the service request is selected randomly from within the following range of values:

- Minimum value: Port number specified in the `dchost` or `dcscdport` operand in the TP1/Client/J environment definition
- Maximum value: Minimum value + number of processes specified in the `dcscdmulticount` operand in the TP1/Client/J environment definition - 1

Note that the value specified in `scdmulti` in the schedule service definition must be the same among all gateway TP1/Servers that are specified in the `dchost` operand in the TP1/Client/J environment definition.

(b) RPCs using the name service

This subsection describes the issuance of RPCs using the name service and multi-scheduler facility. If there is no corresponding service information in the area where the service information is stored temporarily, the service information is queried to the name service. Based on the obtained service information, a multi-scheduler daemon is selected randomly and then a service request is sent.

(2) Relationship between TP1/Client/J environment definition and the scheduler daemon that sends a service request

When the multi-scheduler facility is used, the scheduler daemon that is used to send a service request depends on the specification of the TP1/Client/J environment definition.

The following table shows the relationship between the operand definitions in the TP1/Client/J environment definition and scheduler daemon for the RPCs that use the scheduler direct facility.

Table 2-1: Relationship between operand definitions in the TP1/Client/J environment definition and scheduler daemon (RPCs using the scheduler direct facility)

Operand specifications in the TP1/Client/J environment definition			Scheduler daemon that sends service request
dcscddirect	**dcscdmulti**	**dcscdmulticount**	
Y	Y	S	Multi-scheduler daemon that was selected randomly#
		--	Scheduler daemon that has been started with the port number specified in the `dchost` or `dcscdport` operand in the TP1/Client/J environment definition
	N	Invalid	Scheduler daemon that has been started with the port number specified in the `dchost` or `dcscdport` operand in the TP1/Client/J environment definition

Legend:

Y: Operand value is `Y`.

N: Operand value is `N`.

S: Operand value is specified.

--: Operand value is not specified.

#

The port number for the multi-scheduler daemon is selected from within the following range of values:

Minimum value: Port number specified in the `dchost` or `dcscdport` operand in the TP1/Client/J environment definition

Maximum value: Minimum value + number of processes specified in the `dcscdmulticount` operand in the TP1/Client/J environment definition - 1

The following table shows the relationship between operand definitions in the TP1/Client/J environment definition and the scheduler daemon for the RPCs that use the name service.

Table 2-2: Relationship between operand definitions in the TP1/Client/J environment definition and scheduler daemon (RPCs using the name service)

Operand specifications in the TP1/Client/J environment definition			**Scheduler daemon that sends service request**
`dcnamuse`	**`dcscdmulti`**	**`dcscdmulticount`**	
Y	Y	Invalid	Multi-scheduler daemon selected randomly on the basis of the service information#
	N		Master scheduler daemon#

Legend:

Y: Operand value is `Y`.

N: Operand value is `N`.

#

A query to the name service occurs.

2.3 Transaction control

This facility is applicable only when the remote API facility is used and the version of the request destination TP1/Server Base is 05-00 or later.

You can call a method that controls transactions from a CUP. To do this, you must specify in advance `atomic_update=Y` in the user service definition for an SPP that is executed as a transaction.

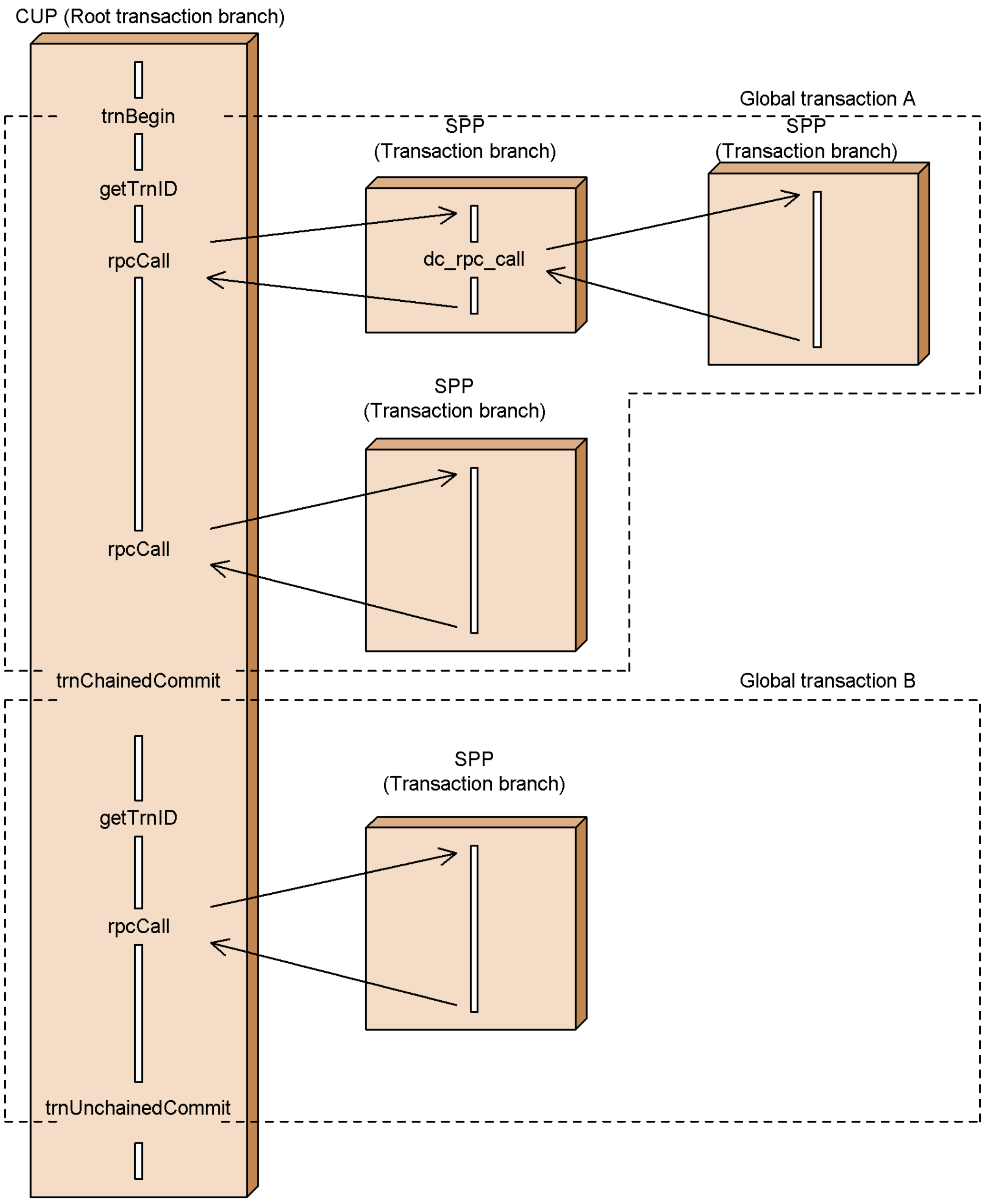
For details about transaction control, see the manual *OpenTP1 Programming Guide*.

2.3.1 Start and synchronization point acquisition of transactions

You can start a transaction by calling the `trnBegin` method from a CUP.

The range of a global transaction is from the time the `trnBegin` method is called to the time a synchronization point is acquired (commit). Once you call the `trnBegin` method, you cannot call another `trnBegin` method within the same global transaction. When an RPC is issued to an SPP from a CUP, the CUP becomes a root transaction branch and the SPP to which the RPC was issued is executed as a transaction branch. The following figure shows the relationships between a transaction and an RPC when `Y` is specified for the `dcrapautoconnect` operand of the TP1/Client/J environment definition.

Figure 2-11: Relationships between a transaction and an RPC (when Y is specified for the dcrapautoconnect operand)



2.3.2 Synchronization point acquisition

(1) Commit

When a transaction terminates normally, a synchronization point (commit) is acquired by calling a commit request method from the CUP. The global transaction terminates normally when all of its transaction branches terminate normally.

(a) Commit in chained and unchained modes

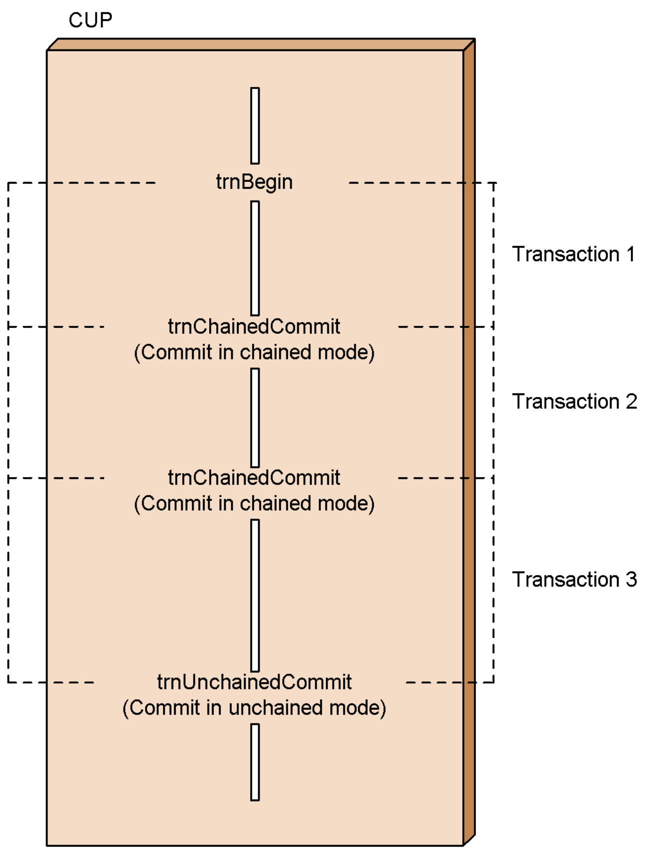
The two modes of synchronization point acquisition for transaction processing are the *commit in chained mode*, in which a synchronization point is acquired after a transaction ends and then the next transaction is started immediately, and the *commit in unchained mode*, in which no new transaction is started after a synchronization point is acquired at the end of a transaction.

To request a commit in the chained mode, call the `trnChainedCommit` method.

To request a commit in the unchained mode, call the `trnUnchainedCommit` method.

The following figure shows the chained and unchained modes of transactions.

Figure 2-12: Chained and unchained modes of transactions



(b) Processing when no commit request method is called

If a CUP terminates without calling a commit request method or a CUP terminates abnormally before calling a commit request method, the corresponding transaction is rolled back.

(2) Rollback

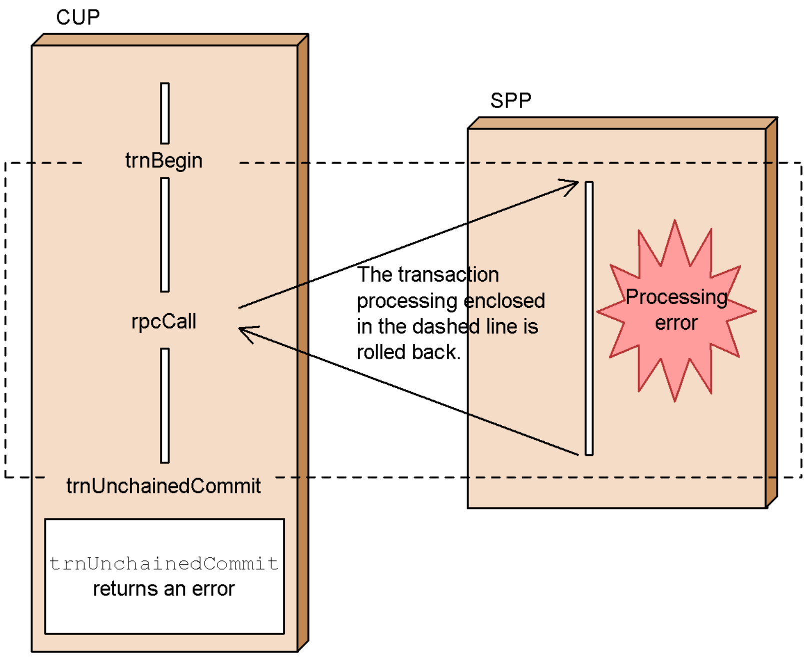
(a) When TP1/Server processing results in an error

If an error occurs in a transaction, the commit request method is returned with an error. The corresponding transaction is rolled back for partial recovery. If an error occurs on one of the transaction branches of a global transaction, the entire global transaction becomes subject to rollback. In such a case, TP1/Server executes partial recovery to roll back the transaction branches.

The following figure shows rollback of a transaction when an error occurs in TP1/

Server processing.

Figure 2-13: Rollback of a transaction (when an error occurs in TP1/Server processing)


CUP
SPP
trnBegin
rpcCall
trnUnchainedCommit
trnUnchainedCommit returns an error
The transaction processing enclosed in the dashed line is rolled back.
Processing error

(b) Calling a rollback request method

To roll back a transaction based on a determination by the CUP, call a rollback request method from the CUP.

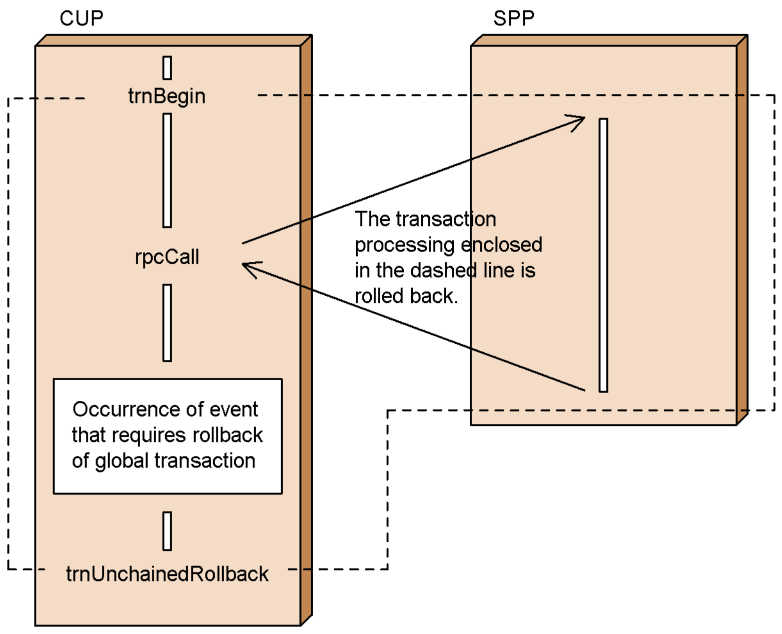
The two modes of rollback are *rollback in chained mode* and *rollback in unchained mode*.

To request a rollback in the chained mode, call the `trnChainedRollback` method. A CUP process that has called the `trnChainedRollback` method for rollback remains in the global transaction range.

To request a rollback in the unchained mode, call the `trnUnchainedRollback` method. A CUP process that has called the `trnUnchainedRollback` method is placed outside the global transaction range after the rollback processing.

The following figure shows rollback of a transaction when a rollback request method is called.

Figure 2-14: Rollback of a transaction (when a rollback request method is called)


CUP
SPP
trnBegin
rpcCall
The transaction processing enclosed in the dashed line is rolled back.
Occurrence of event that requires rollback of global transaction
trnUnchainedRollback

(3) Handling of heuristic situations

If a heuristic situation occurs during transaction processing, an error is returned during CUP synchronization point acquisition. The exceptions are as follows:

- The result of heuristic decision-making does not match the result of the synchronization point of the global transaction: `ErrHeuristicException`
- The result of the synchronization point of the transaction that was heuristically completed is unknown due to an error: `ErrHazardException`

For details about the causes of these exceptions and the synchronization point result for the global transaction, see the TP1/Server's message log file.

For details about how to handle heuristic situations, see the manual *OpenTP1 Programming Guide*.

(4) Processing time of transactions

You can specify the following transaction-related times in the TP1/Client/J environment definition; for details, see *5.2 Details of TP1/Client/J environment definitions*:

- Expiry time in transaction branch

- Whether or not the transaction branch monitoring interval is to include the time needed by the transaction branch being monitored to call another transaction branch using the RPC facility and wait for completion of its processing
- Maximum time of transaction inquiry interval
- Transaction branch CPU monitoring interval

(5) Types of statistical information to be acquired for transaction branches

You can specify in the TP1/Client/J environment definition the types of transaction statistical information that are to be acquired for each transaction branch. For details, see *5.2 Details of TP1/Client/J environment definitions*.

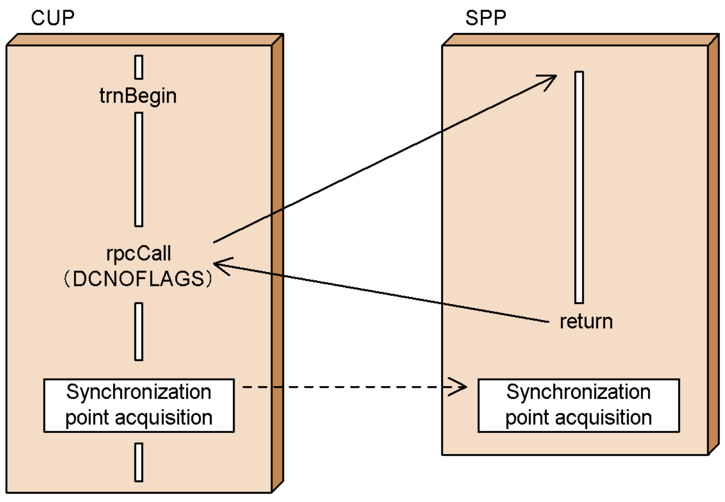
2.3.3 Relationship between mode of remote procedure calls and synchronization point

(1) Relationship between a synchronous-response RPC and the synchronization point

A transaction with a synchronous-response RPC ends when the processing result is returned to the CUP and synchronization point processing is completed.

The following figure shows the relationship between a synchronous-response RPC and the synchronization point.

Figure 2-15: Relationship between a synchronous-response RPC and the synchronization point


CUP
SPP
trnBegin
rpcCall
（DCNOFLAGS）
return
Synchronization point acquisition
Synchronization point acquisition

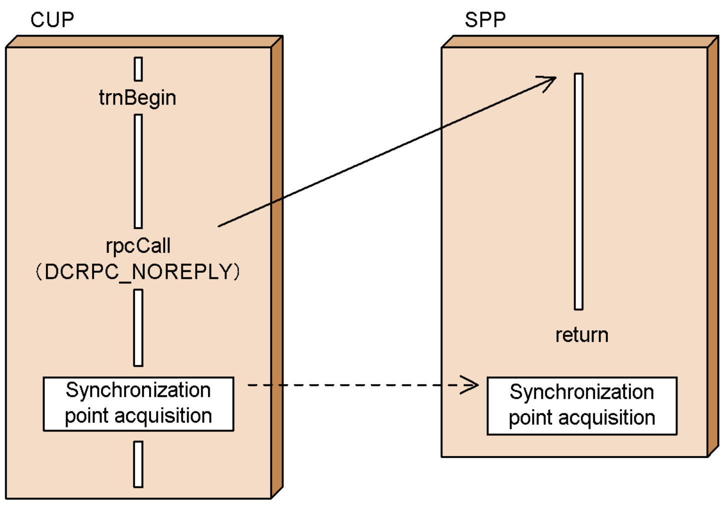
(2) Relationship between a non-response type RPC and the synchronization point

A transaction with a non-response type RPC is synchronized at the end of the CUP and

SPP processing.

The following figure shows the relationship between a non-response type RPC and the synchronization point.

Figure 2-16: Relationship between a non-response type RPC and the synchronization point


CUP
SPP
trnBegin
rpcCall
(DCRPC_NOREPLY)
return
Synchronization point acquisition
Synchronization point acquisition

2.3.4 Acquiring the current transaction identifiers

By calling the `getTrnID` method from a CUP, you can obtain the current transaction global identifier and transaction branch identifier.

In the event of an error, you need the transaction global identifier to determine whether or not the transaction started from the CUP has been committed. To protect against possible errors, you should always call the `getTrnID` method after calling any of the following methods:

- `trnBegin` method
- `trnChainedCommit` method
- `trnChainedRollback` method

2.3.5 Reporting information about the current transaction

By calling the `trnInfo` method from a CUP, you can determine from its return value whether or not a `TP1Client` object is running as a transaction.

2.3.6 Verifying the transaction synchronization point in the event of an error

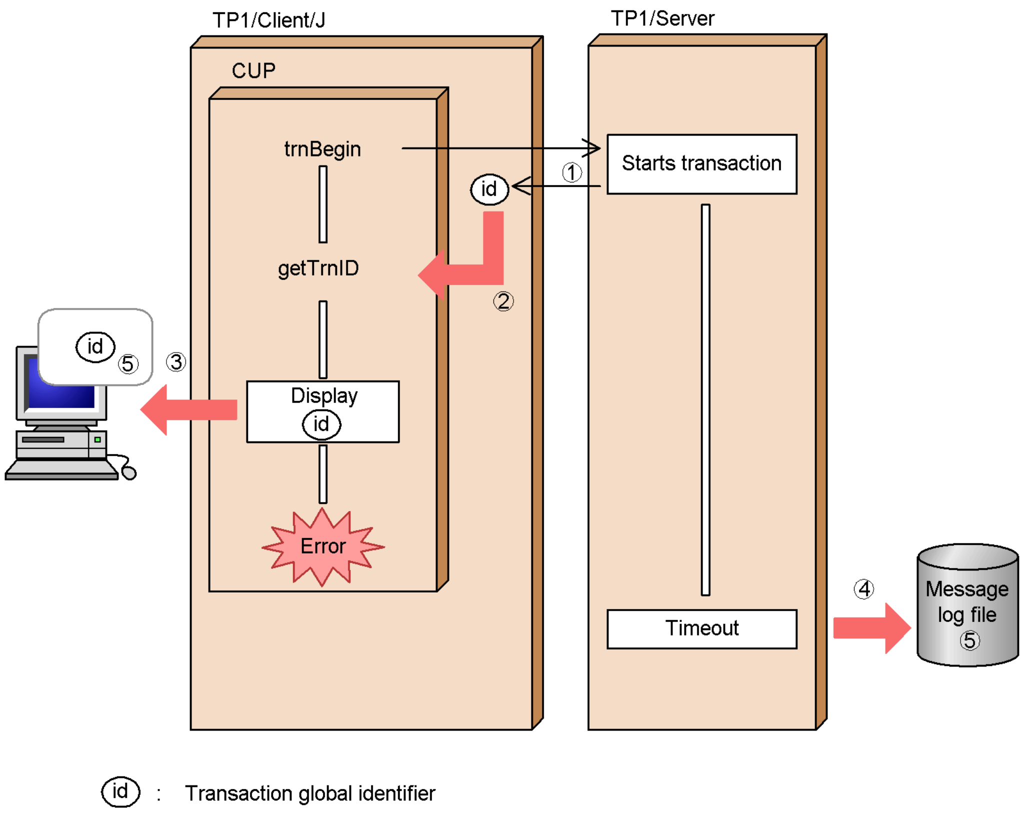
If an error occurs on a transaction that was started by a CUP, you can verify whether or not its transaction branch has been committed. To do this, you must obtain the current transaction global identifier and transaction branch identifier by calling the `getTrnID` method before the transaction is started.

You can determine whether or not the transaction started by the CUP has been committed by comparing the transaction global identifier obtained from the CUP beforehand with the transaction result output to TP1/Server's message log file.

You use the `logcat` command to display the contents of TP1/Server's message log file. For details about the `logcat` command, see the manual *OpenTP1 Operation*.

The following figure shows the method for verifying the transaction synchronization point in the event of an error.

Figure 2-17: Method for verifying the transaction synchronization point in the event of an error



Explanation:

① When a transaction is started by the `trnBegin` method called from a CUP, TP1/Server reports the transaction global identifier of this transaction to TP1/Client/J.

② The CUP calls the `getTrnID` method to acquire the transaction global identifier.

③ The acquired transaction global identifier is displayed.

④ If an error occurs in the CUP resulting in a timeout at the TP1/Server, the processing results of the transaction are output to TP1/Server's message log file.

⑤ Verify the transaction global identifier that was displayed in ③ and the contents of the message log file.

2.3.7 Definitions for TP1/Server

You should note the following points about specifying the `rpc_extend_function` operand in the TP1/Server definitions in order to link transactions between TP1/Client/J and TP1/Server:

- For the `rpc_extend_function` operand in the user service default definition, do not set bit 00000002 to on. If bit 00000002 is on, operations cannot be guaranteed.

- If bit 00000002 in the `rpc_extend_function` operand in the user service default definition is on, define the `rpc_extend_function` operand in the RAP-processing listener service definition so that bit 00000002 is turned off, and use the `rapdfgen` command to re-create the user service definitions for the RAP-processing listener and RAP-processing server. After re-creating the definitions, restart the RAP-processing listener and RAP-processing server.

2.4 TCP/IP communication facility

The communication method used by the TCP/IP communication facility provided by TP1/Client/J differs depending on whether the CUP is operating as a client-type or a server-type program. If the CUP is operating as a client-type program, the TCP/IP communication facility establishes a connection from the CUP to the remote system as specified by definitions and parameters. If the CUP is operating as a server-type program, the facility waits for a remote system to establish a connection on the port that is defined in the definitions before initiating communication. Once a connection has been established, the facility sends and receives data in the area specified by the parameter.

Note:

Use of the TCP/IP communication facility enables communication with MHPs. In this manual, the remote system is referred to as an MHP, but the user is free to select anything, not only an MHP.

Communication of messages using the TCP/IP communication facility can take any of the following three forms:

- Unidirectional sending of messages from a CUP to an MHP
- Unidirectional receiving of messages from an MHP to a CUP
- Bidirectional sending and receiving of messages between an MHP and a CUP

2.4.1 Unidirectional sending of messages

Messages can be unilaterally sent from a CUP to an MHP. This is called *unidirectional sending of messages*.

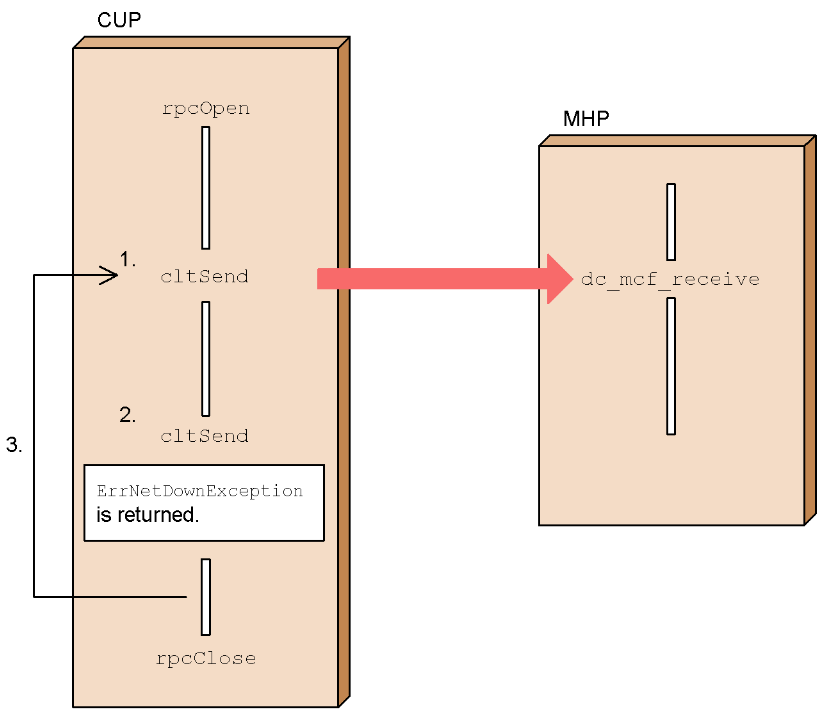
A CUP executes the `cltSend` method to send a message to an MHP.

To enable unidirectional sending of messages, you must preset the following specifications in the TP1/Client/J environment definition:

- Specify the host name of the node on which the MHP resides in the `dcsndhost` operand.
- Specify the port number of the MHP in the `dcsndport` operand (this is the port number specified with the `portno` operand of the `mcftalccn` definition command in the MCF communication configuration definition).
- Specify `DCCLT_ONEWAY_SND` in the `dcsndrcvtype` operand.

The following figure shows unidirectional sending of a message.

Figure 2-18: Unidirectional sending of a message



Explanation:
1. After the MHP starts, the CUP starts and executes the `cltSend` method.
2. If the connection from the MHP is dropped after the `cltSend` method is issued, the exception `ErrNetDownException` is returned to the CUP.
3. To perform unidirectional sending of a message again, re-execute the `cltSend` method.

2.4.2 Unidirectional receiving of messages

A CUP can receive messages sent from an MHP. This is called *unidirectional receiving of messages*.

To use the TCP/IP protocol to receive a message from an MHP, the CUP executes the `cltReceive` method.

The TCP/IP protocol separates each message into multiple packets and packs multiple messages into a single packet. This means that the end of the received message must be determined based on the message length the user specifies. A user initially receives a fixed-length header that includes the message length, and then specifies the message length included in the header to receive the actual message.

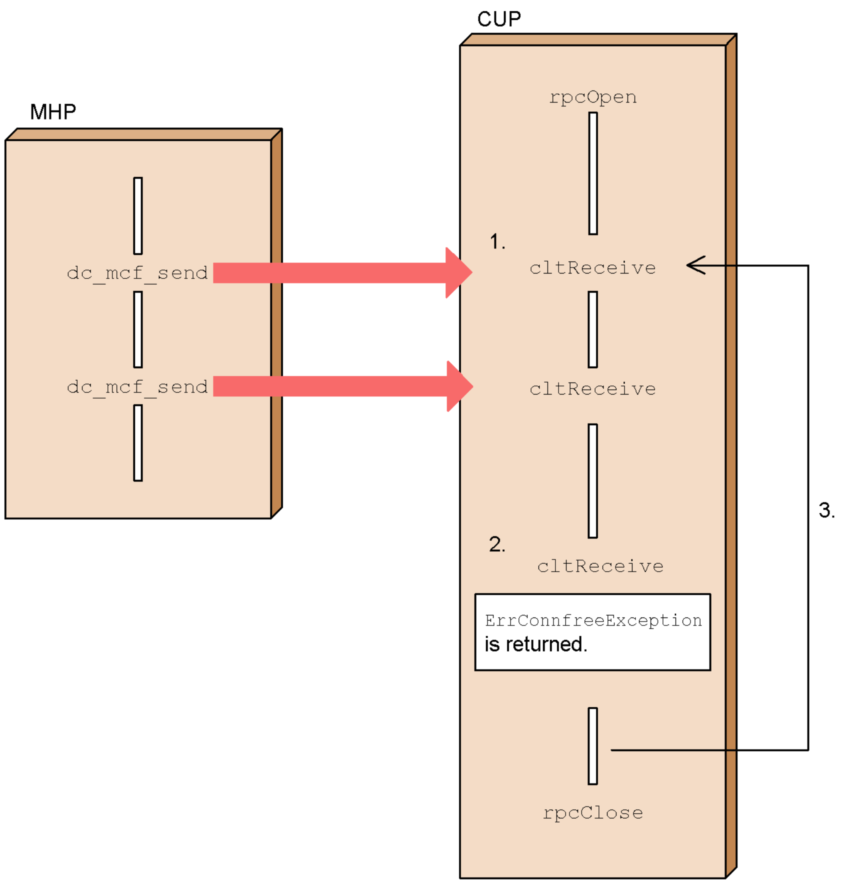
When a message whose length is less than the specified message length is received, TP1/Client/J assumes that the message has been segmented, and does not return control to the CUP until it receives the portion of the message that remains as specified by the message length. If a timeout or error occurs, the message can be recovered up to the point of the error, even if the length of the message received is less than the specified message length. However, the user is responsible for assembly of the message from that point.

To enable unidirectional receiving of messages, you must preset the following specifications in the TP1/Client/J environment definition:

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

The following figure shows unidirectional message reception.

Figure 2-19: Unidirectional receiving of a message

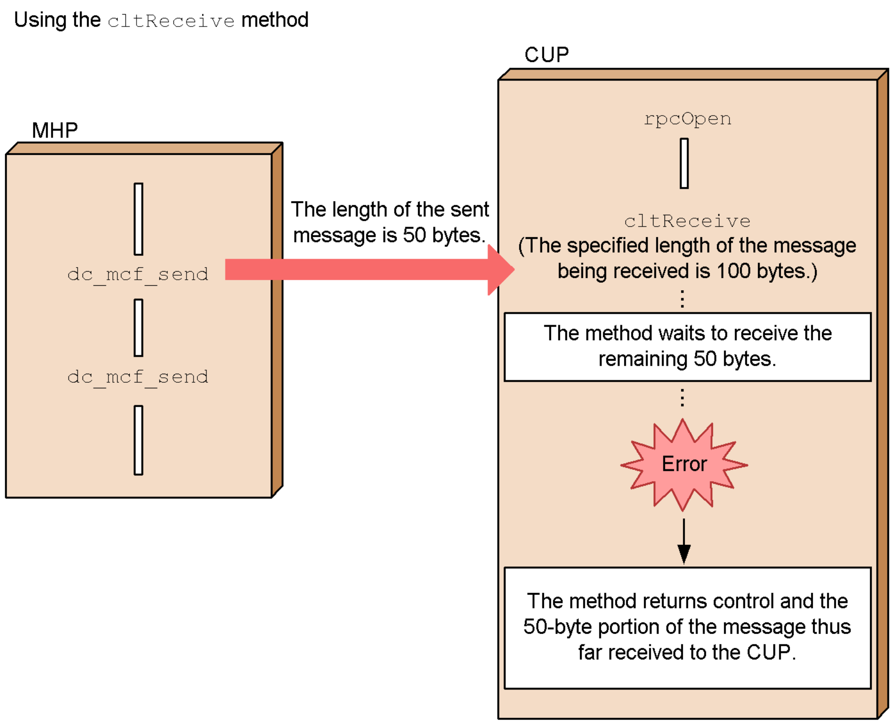


Explanation:

1. While the MHP is retrying connection establishment requests, the CUP is started and executes the `cltReceive` method. If the MHP cannot establish a connection during the retries, the `mcftactcn` command can be used to establish a connection.
2. If the connection is dropped by the MHP, the exception `ErrConnFreeException` is returned to the CUP
3. To perform unidirectional receiving of the message once more, the `cltReceive` method can be executed. In this case, enter the `mcftactcn` command to establish the connection.

The following figure shows unidirectional message reception in the event of a failure.

Figure 2-20: Unidirectional receiving of a message when an error occurs


Using the cltReceive method
CUP
MHP
rpcOpen
dc_mcf_send
dc_mcf_send
The length of the sent message is 50 bytes.
cltReceive
(The specified length of the message being received is 100 bytes.)
The method waits to receive the remaining 50 bytes.
Error
The method returns control and the 50-byte portion of the message thus far received to the CUP.

2.4.3 Bidirectional sending and receiving of messages

Messages can be bidirectionally sent and received between a CUP and an MHP.

The CUP executes a `cltSend` method to send messages to an MHP, and the CUP executes a `cltReceive` method to receive messages from an MHP.

To enable bidirectional sending and receiving of messages, you must preset the following specifications in the TP1/Client/J environment definition:

When MHP is a server

- Specify in the `dcsndhost` operand the host name of the node that contains the MHP.
- Specify in the `dcsndport` operand the port number of the MHP (port number specified in the `portno` operand of the `mcftalccn` definition command in the MCF communication configuration definition).
- Specify `DCCLT_SNDRCV` in the `dcsndrcvtype` operand.

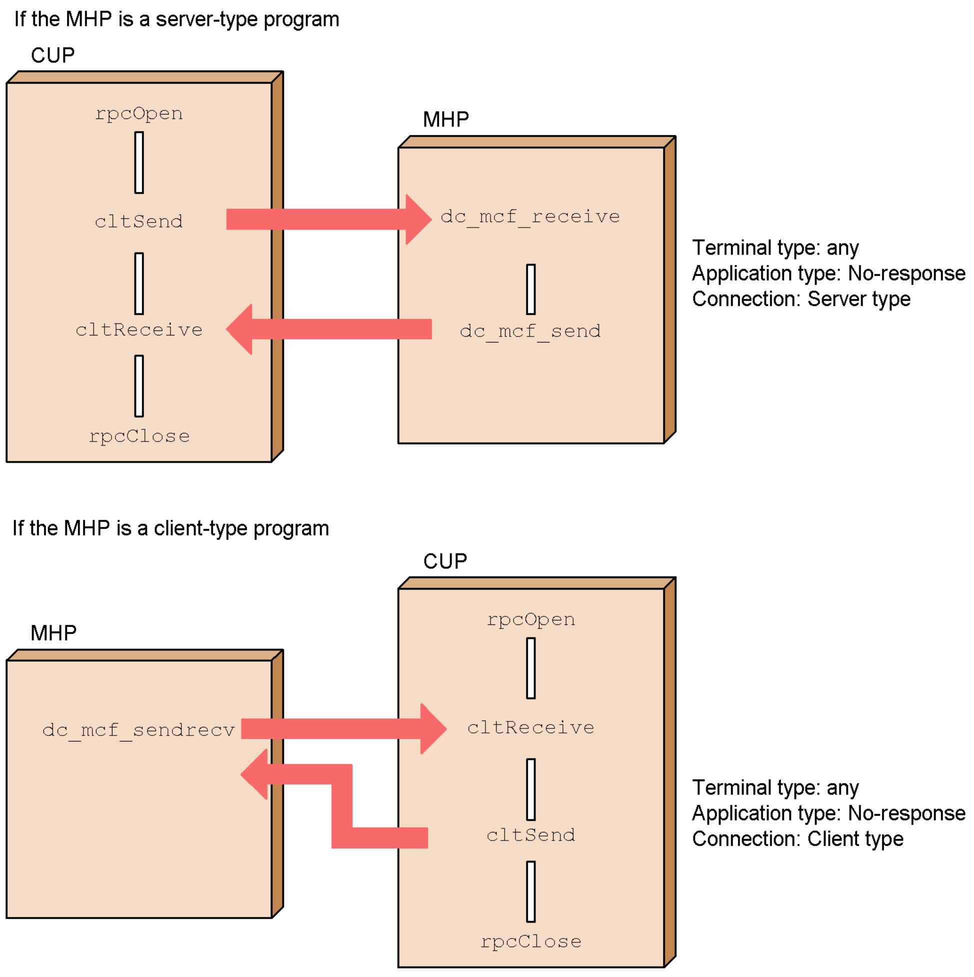
When MHP is a client

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

Although unidirectional sending of messages and unidirectional receiving of messages can be used to send and receive messages by operating over separate connections, bidirectional sending and receiving of messages sends and receives messages using the same connection.

The following figure shows bidirectional sending and receiving of messages.

Figure 2-21: Bidirectional sending and receiving of messages

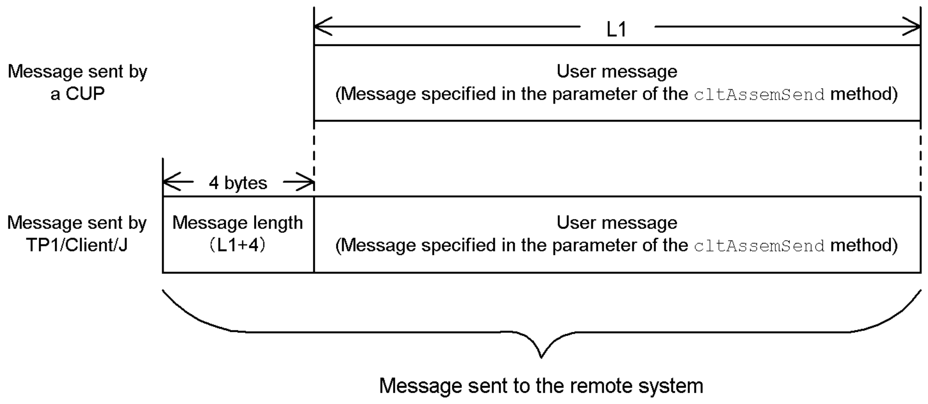


2.4.4 Received message assembly facility

The received message assembly facility carries out TCP/IP communication using the first 4 bytes of a sent or received message as the message length area. If you use this facility when the remote system is using the TP1/NET/TCP/IP received message assembly facility, you can perform communication without being concerned about the message length area in a CUP.

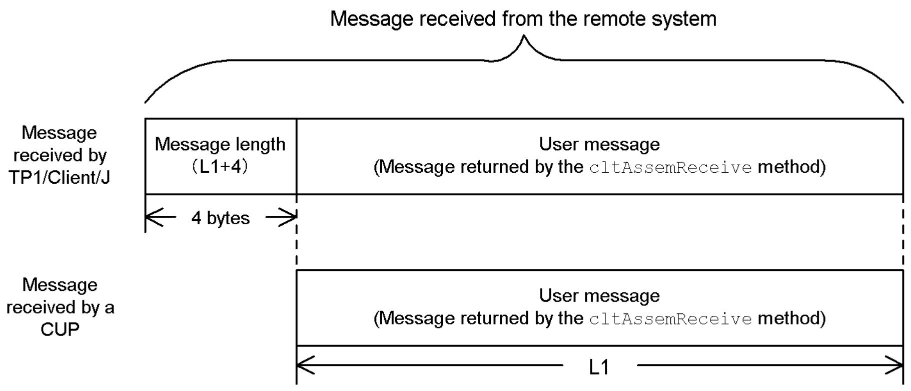
(1) Sending messages

When a CUP uses the `cltAssemSend` method to send a message, TP1/Client/J adds a 4-byte message length area to the beginning of the message and sends it to the remote system. The message length added by TP1/Client/J is set as the network byte order. To use the `cltAssemSend` method, specify either `DCCLT_ONEWAY_SND` or `DCCLT_SNDRCV` in the `dcsndrcvtype` operand of the TP1/Client/J environment definition.


L1
Message sent by a CUP
User message (Message specified in the parameter of the cltAssemSend method)
4 bytes
Message sent by TP1/Client/J
Message length (L1+4)
User message (Message specified in the parameter of the cltAssemSend method)
Message sent to the remote system

(2) Receiving messages

When a CUP uses the `cltAssemReceive` method to make a message-receiving request, TP1/Client/J treats the first 4 bytes of the received message as the message length and assembles or disassembles the message and reports it to the CUP. The value set as the message length is processed as the network byte order. The remote system that sends the message must set the message length as a network byte order. To use the `cltAssemReceive` method, specify either `DCCLT_ONEWAY_RCV` or `DCCLT_SNDRCV` in the `dcsndrcvtype` operand of the TP1/Client/J environment definition.


Message received from the remote system
Message received by TP1/Client/J
Message length (L1+4)
User message (Message returned by the cltAssemReceive method)
4 bytes
Message received by a CUP
User message (Message returned by the cltAssemReceive method)
L1

2.4.5 Notes about using the TCP/IP communication facility

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

(1) Notes on sending messages

(a) Losing a message when an error occurs

If either of the errors described below occurs, TP1/Client/J and the MHP cannot detect that a message has been lost. Therefore, the user must assign a sequential number in the message or use some other means to be prepared for these types of errors:

- Immediately after TP1/Client/J wrote the message to the socket buffer and the transmission ended normally, a communication error occurs or the connection is dropped.
- Immediately before the MHP writes a message sent from TP1/Client/J to its receive buffer, a communication error occurs or the connection is dropped.

(b) Establishing a connection

TP1/Client/J can become a client and send messages to an MHP. This means that TP1/Client/J establishes a connection to the MHP. If the MHP is using TP1/NET/TCP/IP, the connection is a server-type connection.

(2) Notes on receiving of messages

(a) Losing a message when an error occurs

If either of the errors described below occurs, TP1/Client/J and the MHP cannot detect that a message has been lost. Therefore, the user must assign a sequential number in the message or use some other means to be prepared for these types of errors:

- Immediately after the MHP wrote the message to the socket buffer and the transmission ended normally, a communication error occurs or the connection is dropped.

- Immediately before TP1/Client/J writes a message sent from the MHP to its receive buffer, a communication error occurs or the connection is dropped.

(b) Checking received messages

Messages can be received from any MHP. Therefore, if a connection establishment request is received, accept the request unconditionally and receive the message. The user must determine if the message is directed to the CUP by checking if the message identifier included in the header is the one included in the message body.

(c) Message length

Messages are received using the TCP/IP protocol. The TCP/IP protocol separates each message into multiple packets and packs multiple messages into a single packet. This means that the end of the received message must be determined based on the message length the user specifies. The user initially receives a fixed-length header that includes the message length, and then specifies the message length included in the header to receive the actual message.

When a message whose length is less than the specified message length is received, TP1/Client/J assumes that the message has been segmented. Therefore, it does not return control to the CUP until it receives the portion of the message that remains as specified by the message length.

(d) Establishing a connection

TP1/Client/J can become a server and receive messages from an MHP. This means that the MHP establishes a connection to TP1/Client/J. If the MHP is using TP1/NET/TCP/IP, the connection is a client-type connection.

2.5 Unidirectional server message reception facility

This section describes the facility for receiving unidirectional messages from server to client.

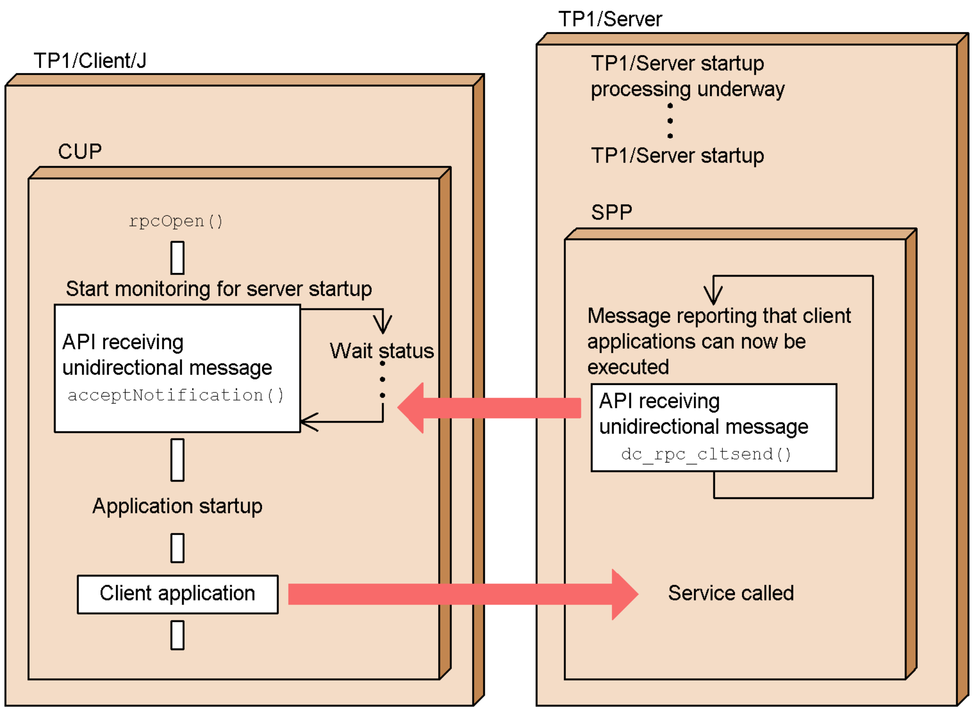
2.5.1 Flow of unidirectional message reception facility processing

The unidirectional message reception facility enables you to use the batch mode to distribute to clients a notification of startup of online operations, which is similar to batch startup of terminals in OLTP in a conventional mainframe system.

To use the unidirectional message reception facility, you execute the `acceptNotification` method. The client waits for a message from the server for the amount of time specified in the method, regardless of the server's status (active or inactive). If a message is sent when the server starts, the client can detect startup of the server and subsequently start user applications (CUPs).

The following figure shows the flow of unidirectional message reception facility processing.

Figure 2-22: Flow of unidirectional message reception facility processing

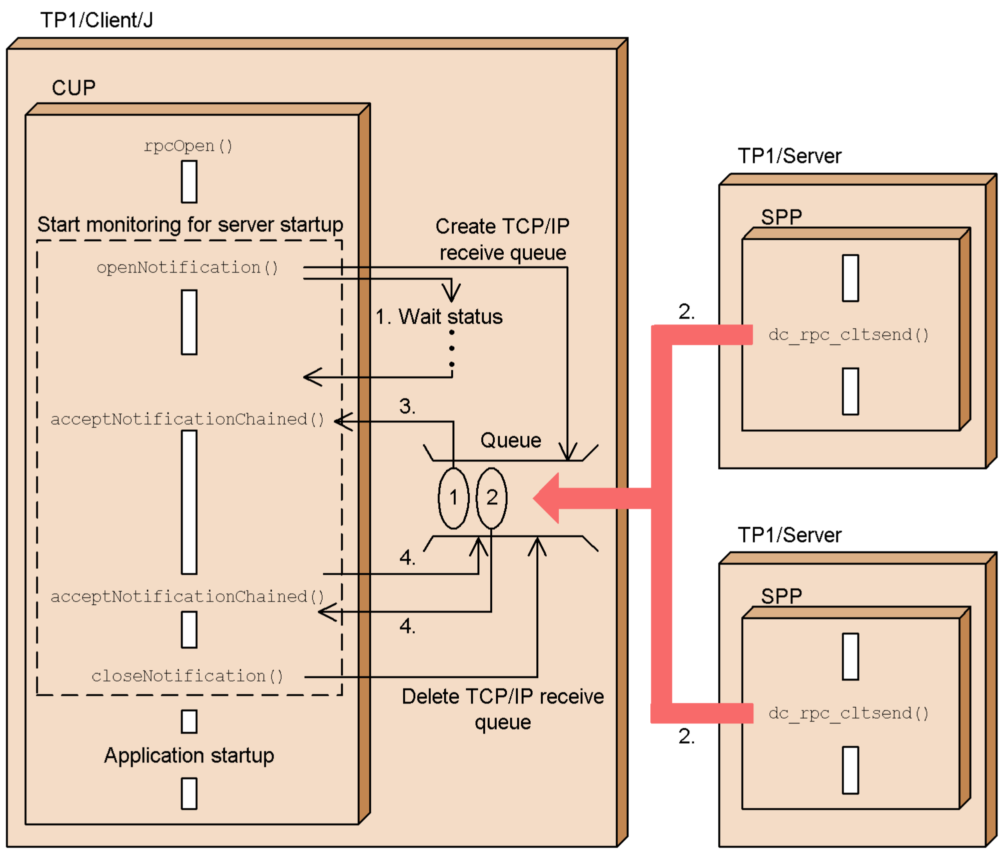

TP1/Client/J
CUP
rpcOpen()
Start monitoring for server startup
API receiving unidirectional message
acceptNotification()
Wait status
Application startup
Client application
TP1/Server
TP1/Server startup processing underway
TP1/Server startup
SPP
Message reporting that client applications can now be executed
API receiving unidirectional message
dc_rpc_cltsend()
Service called

2.5.2 Flow of unidirectional message consecutive reception facility processing

The unidirectional message consecutive reception facility allows you to receive consecutively unidirectional messages from the server after execution of the `openNotification` method; receipt of such messages continues until the `closeNotification` method is executed. If this facility is used but the client is not ready to receive unidirectional messages from the server, transmission of the unidirectional messages from the server does not result in an error. The client can retrieve the messages subsequently from the receive queue by executing the `acceptNotificationChained` method for receiving unidirectional messages.

The following figure shows the flow of unidirectional message consecutive reception facility processing.

Figure 2-23: Flow of unidirectional message consecutive reception facility processing



1. Waits for a unidirectional message.
2. Sends a unidirectional message to notify that TP1/Server has started and client applications can now be executed. The unidirectional message from the server is stored in the TCP/IP receive queue.
3. Retrieves the unidirectional message from the TCP/IP receive queue and returns control to the CUP.
4. Issues the `acceptNotificationChained` method to retrieve from the TCP/IP receive queue the unidirectional message sent from the server, then returns control to the CUP.

2.5.3 Notes about using the unidirectional message consecutive reception facility

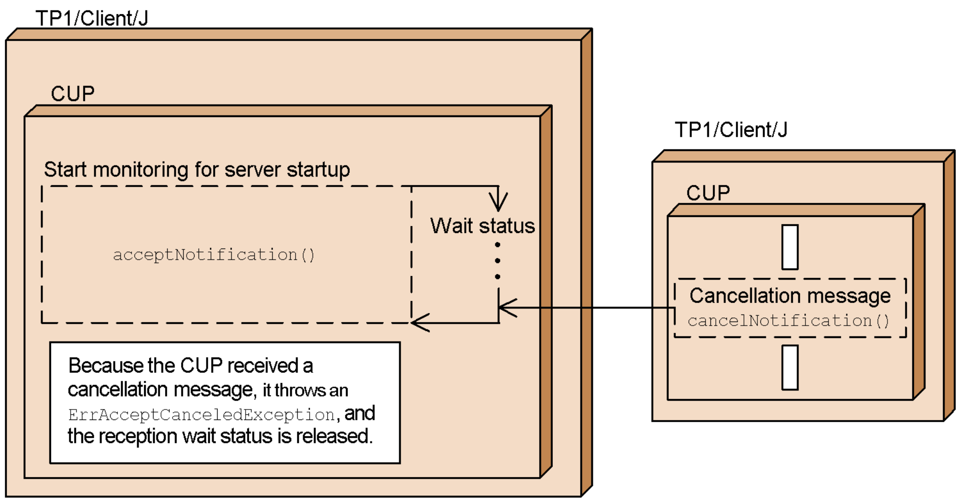
There is a limit to the number of messages that can be retained in the TCP/IP receive queue. This limit depends on a JavaVM maximum value. If the number of messages received exceeds the maximum value, the `dc_rpc_cltsend` function executed at the server results in an error with `DCRPCER_SERVICE_NOT_UP`.

2.5.4 Releasing the unidirectional message reception wait status

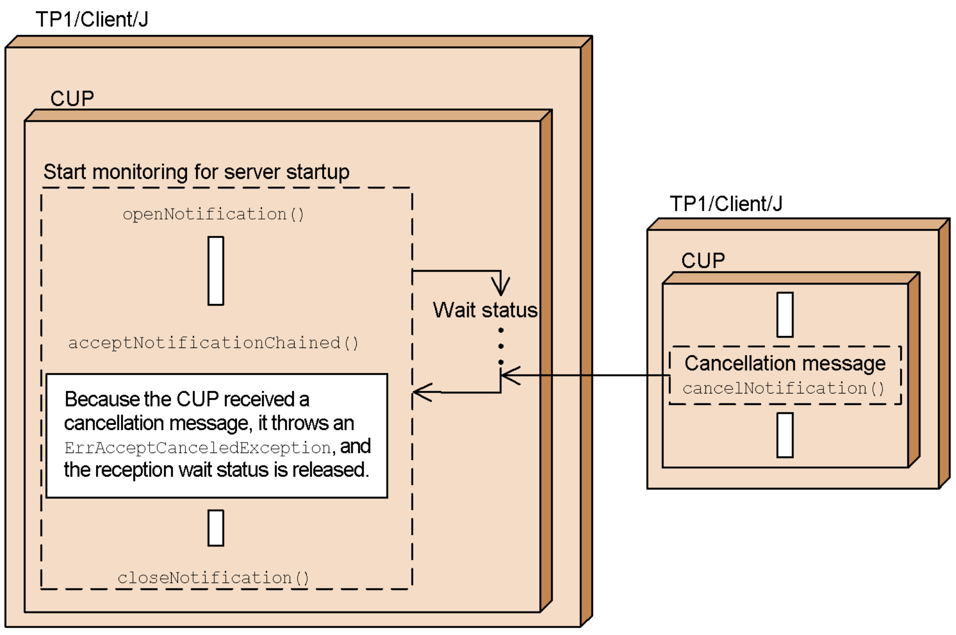
If a CUP issues the `acceptNotification` or `acceptNotificationChained` method and is placed in unidirectional server message reception wait status, and then receives a cancellation message from another CUP, the unidirectional message reception wait status is released. You can send a cancellation message by issuing the `cancelNotification` method. The following figure shows the flow of releasing the unidirectional message reception wait status.

Figure 2-24: Flow of releasing unidirectional message reception wait status

• When unidirectional message reception facility is used


TP1/Client/J
CUP
Start monitoring for server startup
acceptNotification()
Wait status
Because the CUP received a cancellation message, it throws an ErrAcceptCanceledException, and the reception wait status is released.
TP1/Client/J
CUP
Cancellation message
cancelNotification()

• When unidirectional message consecutive reception facility is used


TP1/Client/J
CUP
Start monitoring for server startup
openNotification()
acceptNotificationChained()
Wait status
Because the CUP received a cancellation message, it throws an ErrAcceptCanceledException, and the reception wait status is released.
closeNotification()
TP1/Client/J
CUP
Cancellation message
cancelNotification()

2.6 TP1/Web connection facility

The TP1/Web connection facility provides a communication service based on the HTTP protocol by connecting a CUP created as a Java applet with TP1/Web.

By connecting with TP1/Web, you can perform communication using the HTTP protocol, which allows you to establish links to previously unconnectable sessions, enabling you to establish continuous interactive operations.

You use the `openConnection` method to connect to TP1/Web. For details about the `openConnection` method, see *4. Classes Used with TP1/Client/J*.

However, note that, if you use the TP1/Web connection facility, you cannot use the transaction control facility. When you use the TP1/Web connection facility with the `dccltrpcmaxmsgsize` operand specified in the TP1/Client/J environment definition, an error may occur at the communication destination TP1/Server node.

2.6.1 Connecting to TP1/Web (starting a session)

A connection to TP1/Web (a session) is started by calling the `openConnection(String url, short flags)` method. A session continues until `closeConnection()` is called.

A session can be started with any of the three ways described below:

- By specifying, in the TP1/Client/J environment definition, the URL of the service request destination in the `dcweburl` operand and `Y` in the `dcrapautoconnect` operand, calling the `rpcOpen` method, and then calling the `rpcCall` method. This causes TP1/Client/J to automatically establish a session with TP1/Web.
- By specifying the URL of the service request destination in the `dcweburl` operand of the TP1/Client/J environment definition, calling the `rpcOpen` method, and then calling the `openConnection` method with no parameters.
- By calling the `openConnection(String url, short flags)` method of the TP1/Client/J API.

2.6.2 Service requests to TP1/Web

Service requests to TP1/Web are performed by calling the `rpcCall` method.

If the TP1/Web Java to SPP gateway facility (`DC_JGW` facility) is used, TP1/Web sends the service request to the SPP on the backend OpenTP1 based on the message sent by the `rpcCall` method. TP1/Web answers response messages from the OpenTP1 SPP using the HTTP protocol.

If the TP1/Web Java to User Service facility (`DC_JUSR` facility) is used, user services running on TP1/Web can be called. You can freely describe processes with the user services running on TP1/Web. TP1/Web returns responses using the HTTP protocol.

If the TP1/Web connection facility provided by TP1/Client/J is used, there are two service request modes: synchronous response mode and chained mode. However, the chained mode uses the TP1/Web static session schedule facility, which can be used only when the `DC_JGW` facility is being used.

For details about the synchronous response mode and the chained mode, see *2.2.3 RPC modes*.

(1) Synchronous response mode

In this mode, a CUP (Java applet) that uses TP1/Client/J sends a query message to TP1/Web and receives a response message. With synchronous response mode RPC, the next process does not execute until the CUP receives the response from TP1/Web.

(2) Chained mode

In this mode, a CUP (Java applet) that uses TP1/Client/J sends a query message to TP1/Web and receives a response message. With chained mode RPC, the next process does not execute until the CUP receives the response from TP1/Web. To use chained mode RPC, TP1/Web must be set to use the static session schedule facility, and the TP1/Web service set requested by the `openConnection` method must be `DC_JGW`.

2.6.3 Releasing a connection to TP1/Web (ending a session)

A connection to TP1/Web is released by calling the `closeConnection` method.

The TP1/Web process that receives a `closeConnection` method ends the session with the connected client and releases TP1/Web resources.

2.6.4 RPC facilities for connecting to TP1/Web

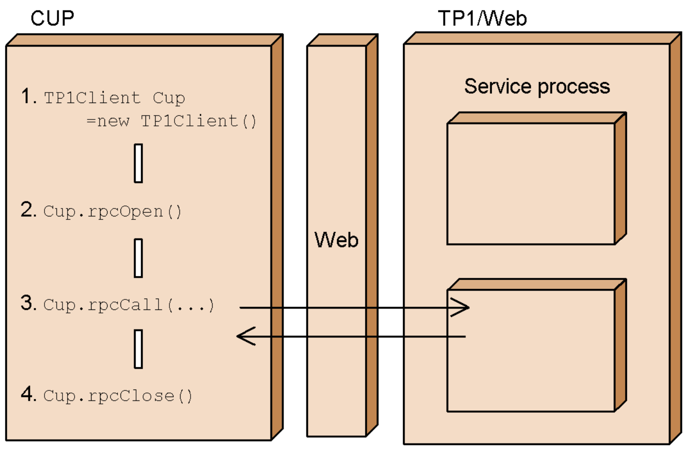
The following lists the TP1/Web service sets that can be used from TP1/Client/J:

- Java to SPP gateway facility (`DC_JGW` facility)
- Java to user service facility (`DC_JUSR` facility)

For details about these service sets, see the manual *OpenTP1 TP1/Web User's Guide*.

Figures 2-25 to 2-29 show the flow of service requests issued by the RPC facility linked with TP1/Web.

Figure 2-25: Connecting by specifying dcrapautoconnect=Y

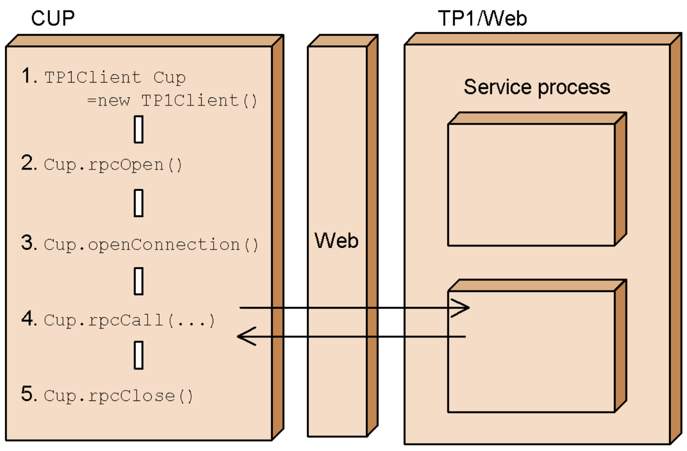


Explanation:

The following operation flow occurs if `dcrapautoconnect=Y` is specified:

1. An instance of the TP1Client class provided by TP1/Client/J is created.
2. The `rpcOpen` method is called to initialize the CUP's RPC environment.
3. The `rpcCall` method is called, which sends a service request to the desired SPP.
   Internal processing in the `rpcCall` method is as follows:
   1. Because `dcrapautoconnect=Y`, start a session with the TP1/Web CGI process defined in the `dcweburl` operand.
   2. Send a service request message.
4. The `rpcClose` method is called, which releases the RPC environment (ends the session).

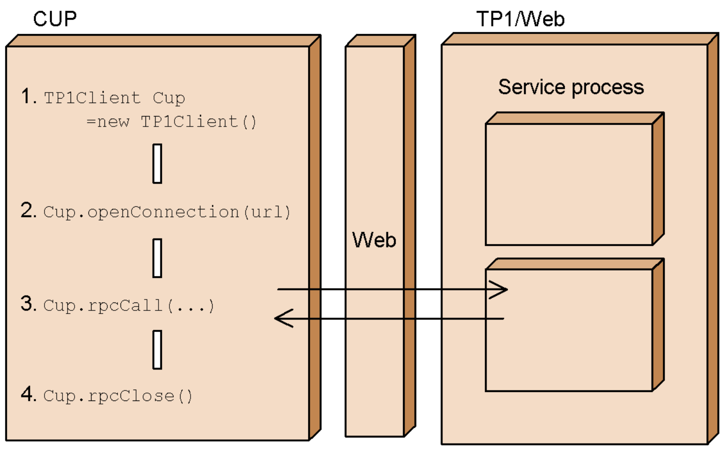
Figure 2-26: Connecting by specifying dcrapautoconnect=N and defining the dcweburl operand



Explanation:
The following operation flow occurs if `dcrapautoconnect=N` is specified and the `dcweburl` operand is defined:

1. An instance of the TP1Client class provided by TP1/Client/J is created.
2. The `rpcOpen` method is called to initialize the CUP's RPC environment.
3. The `openConnection` method is called without any parameters, and a session is started with the TP1/Web CGI process defined in the `dcweburl` operand.
4. The `rpcCall` method is called, which sends a service request to the desired SPP.
5. The `rpcClose` method is called, which releases the RPC environment (ends the session).

Figure 2-27: Connecting by using an openConnection method in which a URL is specified



Explanation:
The following operation flow occurs if an `openConnection` method in which a URL is specified is used:

1. An instance of the TP1Client class provided by TP1/Client/J is created.
2. The `openConnection` method specifying the URL is called to start a session with the TP1/Web CGI process.
3. The `rpcCall` method is called, which sends a service request to the desired SPP.
4. The `rpcClose` method is called, which releases the RPC environment (ends the session).

Figure 2-28: Flow of service execution when using the DC_JGW service set

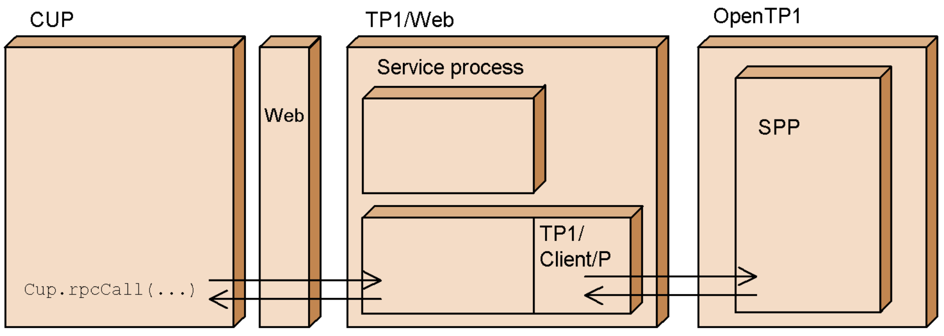
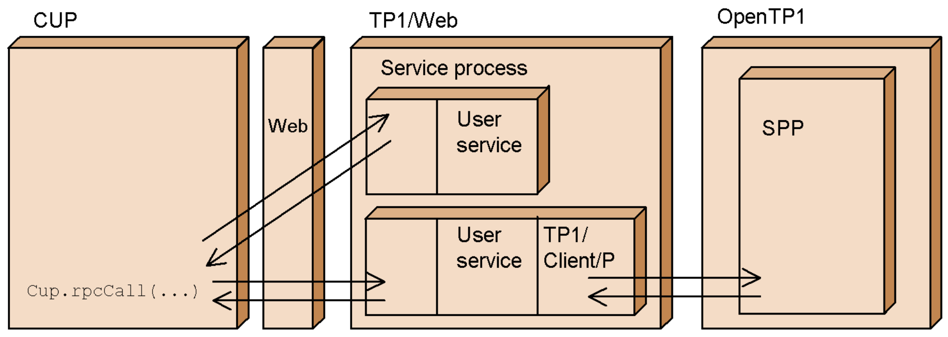


Figure 2-29: Flow of service execution when using the DC_JUSR service set



2.7 Dynamic definition changing facility

TP1/Client/J analyzes the TP1/Client/J environment definition within the `rpcOpen` method and keeps it as unique information in the CUP.

The dynamic definition changing facility enables you to specify operands that are not currently defined in the TP1/Client/J environment definition and to change dynamically operand values that are specified in the TP1/Client/J environment definition. Once you change an operand value by calling the dynamic definition changing method, the new value remains in effect until the CUP is terminated (until the `rpcClose` method is called) unless the dynamic definition changing method is called to change the same operand value again.

Whether the value in the TP1/Client/J environment definition or the value set by the dynamic definition changing facility is in effect depends on the order in which the methods are called.

When the dynamic definition changing method is called after the `rpcOpen` method

> The operand value changed by the dynamic definition changing method is in effect.

When the `rpcOpen` method is called after the dynamic definition changing method

> The operand value defined in the TP1/Client/J environment definition is in effect.

2.8 Facility for monitoring TCP/IP connection establishment

In the facility for monitoring TCP/IP connection establishment, you can specify the maximum amount of time to monitor for the establishment of a TCP/IP connection for sending data, and monitor for connection establishment.

To establish connection, the `java.net.Socket.connect` method (hereafter referred to as *connect method*), which is a JavaAPI inside the API of TP1/Client/J, is used. The time-out monitoring time for the `connect` method is uniquely set in each platform. By monitoring connection establishment by specifying the maximum monitoring time in the TP1/Client/J environment definition, you can make the API of TP1/Client/J return an error earlier than the `connect` method time-out monitoring time unique to the platform. You specify the maximum monitoring time for connection establishment in the `connect` method using the `dccltconnecttimeout` operand of the TP1/Client/J environment definition.

2.8.1 API that uses the connect method internally

Whether an API uses the `connect` method internally depends on the operand specification in the TP1/Client/J environment definition. The table below shows the APIs (methods) that internally use the `connect` method and the conditions for the operands of the TP1/Client/J environment definition inside which the `connect` method is used. If two or more conditions are listed for an API (method), at least one of the conditions must be satisfied for the `connect` method to be used internally.

Table 2-3: APIs (methods) that use the connect method internally

Item No.	Methods that use the connect method internally	Conditions for the operands of the TP1/Client/J environment definition inside which the connect method is used
1	`cltAssemSend(byte[] buff, int sendleng, String hostname, int portnum, int timeout, int flags)`	• `dcsndrcvtype=DCCLT_ONEWAY_SND` • `dcsndrcvtype=DCCLT_SNDRCV`
2	`cltSend(byte[] buff, int sendleng, String hostname, int portnum, int flags)`	• `dcsndrcvtype=DCCLT_ONEWAY_SND` • `dcsndrcvtype=DCCLT_SNDRCV`
3	`openConnection(String host, int port)`	`dcrapdirect=Y` and `dcrapautoconnect=N`
4	`openConnection()`	`dcrapdirect=Y` and `dcrapautoconnect=N`

Item No.	Methods that use the connect method internally	Conditions for the operands of the TP1/Client/J environment definition inside which the connect method is used
5	`rpcCall(String group, String service, byte[] in_data, int[] in_len, byte[] out_data, int[] out_len, int flags)`	• `dcscddirect=Y` • `dcnamuse=Y` or • `dcrapdirect=Y` and `dcrapautoconnect=Y`
6	`rpcCall(String group, String service, byte[] in_data, byte[] out_data, int flags)`	
7	`rpcCallTo(DCRpcBindTbl direction, String group, String service, byte[] in_data, int[] in_len, byte[] out_data, int[] out_len, int flags)`	• `dcscddirect=Y` • `dcnamuse=Y`
8	`trnBegin()`	`dcrapdirect=Y` and `dcrapautoconnect=Y`

2.8.2 Exceptions that occur when the connect method times out

The API of TP1/Client/J returns different exceptions depending on whether the `connect` method time-out monitoring time unique to each platform or the time specified in the `dccltconnecttimeout` operand expires. This is because the difference in the monitoring time that expires affects the exceptions that occur in the `connect` method.

If a value greater than the time-out monitoring time unique to each platform is specified in the `dccltconnecttimeout` operand, the time-out monitoring time unique to each platform expires first. Consequently, the specification of the `dccltconnecttimeout` operand becomes invalid. If the specification of the `dccltconnecttimeout` operand is omitted, the `connect` method time-out monitoring time becomes unique to each platform.

The following table shows the exceptions that the API of TP1/Client/J returns when the `connect` method time-out monitoring time unique to each platform expires or when the time specified in the `dccltconnecttimeout` operand expires.

Table 2-4: Exceptions returned by the API (method) when the connect method times out

Item No.	Method	Unique to each platform[#1]	dccltconnecttimeout operand[#2]
1	`cltAssemSend(byte[] buff, int sendleng, String hostname, int portnum, int timeout, int flags)`	`ErrNetDownAtClientException`	`ErrClientTimedOutException`
2	`cltSend(byte[] buff, int sendleng, String hostname, int portnum, int flags)`	`ErrNetDownAtClientException`	`ErrClientTimedOutException`
3	`openConnection(String host, int port)`	`ErrNetDownAtClientException`	`ErrClientTimedOutException`
4	`openConnection()`	`ErrNetDownAtClientException`	`ErrClientTimedOutException`
5	`rpcCall(String group, String service, byte[] in_data, int[] in_len, byte[] out_data, int[] out_len, int flags)`	`ErrNetDownAtClientException`	`ErrClientTimedOutException`
6	`rpcCall(String group, String service, byte[] in_data, byte[] out_data, int flags)`	`ErrNetDownAtClientException`	`ErrClientTimedOutException`
7	`rpcCallTo(DCRpcBindTbl direction, String group, String service, byte[] in_data, int[] in_len, byte[] out_data, int[] out_len, int flags)`	`ErrConnRefusedException`	`ErrNetDownAtClientException`
8	`trnBegin()`	`ErrConnRefusedException`	`ErrClientTimedOutException`

#1

Exceptions that occur when the `connect` method time-out monitoring time unique to each platform expires.

#2

Exceptions that occur when the time specified in the `dccltconnecttimeout` operand expires.

2.9 DCCM3 connection facility

TP1/Client/J can use an RPC or a TCP/IP communication facility to connect to DCCM3 on VOS3 or VOS1.

2.9.1 Issuing an ROC to DCCM3

TP1/Client/J can use an RPC to communicate with an OpenTP1 server as well as with a DCCM3 server. To use an RPC to communicate with a DCCM3 server, a function for interpreting an OpenTP1 RPC must be installed on the DCCM3 server. When the following products are installed on the DCCM3 server, you can use an RPC to communicate with the DCCM3 server.

When the OS on the DCCM3 server is VOS3:

DCCM3/Internet

When the OS on the DCCM3 server is VOS1:

DCCM3/SERVER/TP1

Note the following when the remote server is DCCM3:

- The RPC modes that can be used are synchronous-response RPC and non-response type RPC. Chained RPC cannot be used.
- The transaction control facility cannot be used.
- When using a permanent connection to issue an RPC to a DCCM3 logical terminal, you can use the load distribution facility. For details, see *2.9.1(4) Load distribution when issuing an RPC to a DCCM3 logical terminal*.
- When an RPC is issued to DCCM3, the service name is evaluated as the transaction name.

(1) Specifying the remote server

To use an RPC to communicate with a DCCM3 server, use a service group name and a service name to specify the remote server. This specification method is the same as that used when issuing an RPC to an OpenTP1 server.

- Service group name

  For the service group name, specify a valid dummy character string. Specify an identifier consisting of between 1 and 31 characters.

- Service name

  Specify the transaction name of the DCCM3 server. The characters that can be used are letters (A-Z and a-z) and numbers (0-9). The total number of characters must be between 1 and 8.

(2) Defining an address of the remote server

When an RPC is used to communicate with a DCCM3 server, a server not managed by the OpenTP1 name service must be called. Therefore, TP1/Client/J must define server addresses separately for individual service names. To define a server address, specify the host name and port number of the RPC-accepting gateway in the `dchost` operand of the TP1/Client/J environment definition.

(3) Executing the RPC

After defining the address of the remote server, execute the RPC. The procedure for executing the RPC differs depending on the specification content of the TP1/Client/J environment definition as shown below.

(a) When Y is specified for the dcrapautoconnect operand

1. Execute the `rpcOpen` method and load the definition.
2. Execute the `rpcCall` method.

   Connection is established at the RPC-accepting gateway specified in the `dchost` operand of the TP1/Client/J environment definition. After the connection is established, the RPC is issued.

(b) When N is specified for the dcrapautoconnect operand or when the specification is omitted

1. Execute the `rpcOpen` method and load the definition.
2. Execute the `openConnection` method without any arguments.

   Connection is established at the RPC-accepting gateway specified in the `dchost` operand of the TP1/Client/J environment definition.
3. After the connection is established, execute the `rpcCall` method to issue the RPC.

(c) To communicate with a server not specified in the dchost operand

1. Execute the `rpcOpen` method and load the definition.

   This step may be omitted. Even if it is omitted, steps 2 and 3 can be executed.
2. Execute the `openConnection` method with the RPC-accepting gateway (host name and port number) specified in its arguments.
3. After the connection is established, execute the `rpcCall` method to issue the RPC.

(4) Load distribution when issuing an RPC to a DCCM3 logical terminal

When TP1/Client/J uses permanent connection to issue an RPC to DCCM3 logical terminals, the connection targets can be allocated to multiple DCCM3 servers during connection establishment to distribute load. From the host names and port numbers of the multiple DCCM3 logical terminals specified in the `dchost` operand of the TP1/Client/J environment definition, TP1/Client/J randomly selects a connection target and attempts a connection. If TP1/Client/J fails to connect to a DCCM3 logical terminal, it randomly selects another DCCM3 logical terminal and attempts a connection. This process is repeated, and only after all attempts to connect to the DCCM3 logical terminals specified in the `dchost` operand of the TP1/Client/J environment definition have failed, is an error detected.

One of the following two API execution procedures can be used to establish communication between TP1/Client/J and a DCCM3 logical terminal:

- Specify the host name and port number of the DCCM3 logical terminal in the `dchost` operand of the TP1/Client/J environment definition and execute the `openConnection` method without any arguments.

  In this case, a permanent connection is used.

- Specify the host name and port number of the DCCM3 logical terminal in the `dchost` operand of the TP1/Client/J environment definition and specify `Y` in the `dchostselect` operand.

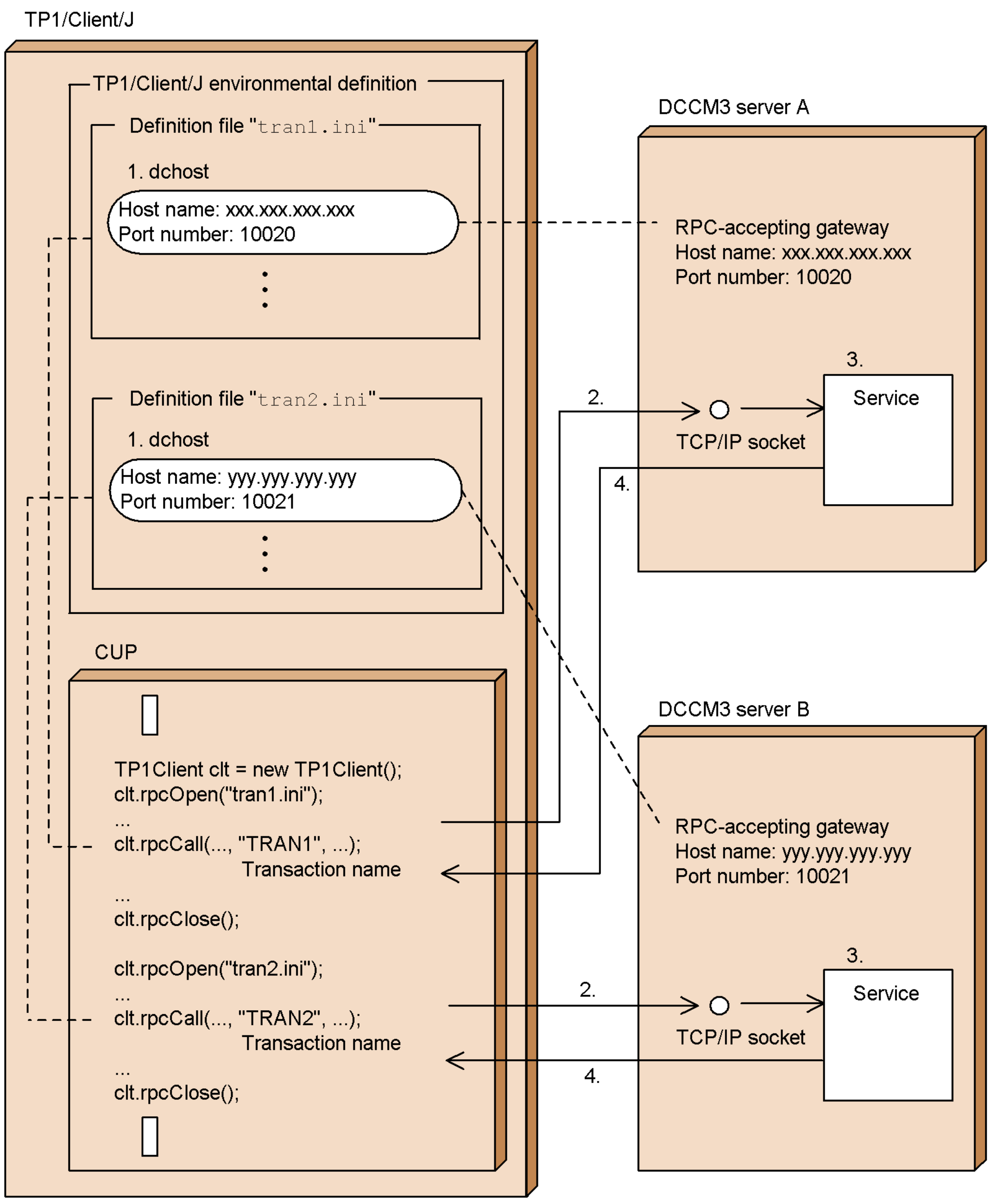
  If `Y` is specified for the `dcrapautoconnect` operand of the TP1/Client/J environment definition, execute the `rpcCall` method. Executing the `rpcCall` method automatically establishes a connection if none has been established.

  If `N` is specified for the `dcrapautoconnect` operand of the TP1/Client/J environment definition or if specification is omitted, execute the `openConnection` method without any arguments. Executing the `openConnection` method without any arguments establishes a connection.

(5) TP1/Client/J environment definition example

The following figure shows an example of TP1/Client/J environment definition used for connecting to a DCCM3 server:

Figure 2-30: Issuing an RPC to a DCCM3 server


TP1/Client/J
TP1/Client/J environmental definition
Definition file "tran1.ini"
1. dchost
Host name: xxx.xxx.xxx.xxx
Port number: 10020
Definition file "tran2.ini"
1. dchost
Host name: yyy.yyy.yyy.yyy
Port number: 10021
CUP
TP1Client clt = new TP1Client();
clt.rpcOpen("tran1.ini");
...
clt.rpcCall(..., "TRAN1", ...);
Transaction name
...
clt.rpcClose();
clt.rpcOpen("tran2.ini");
...
clt.rpcCall(..., "TRAN2", ...);
Transaction name
...
clt.rpcClose();
DCCM3 server A
RPC-accepting gateway
Host name: xxx.xxx.xxx.xxx
Port number: 10020
2.
3.
Service
TCP/IP socket
4.
DCCM3 server B
RPC-accepting gateway
Host name: yyy.yyy.yyy.yyy
Port number: 10021
2.
3.
Service
TCP/IP socket
4.

1. In the TP1/Client/J environment definition, define in separate definition files the addresses (RPC-accepting gateways) of the servers that process transactions in the `dchost` operand.

2. When an RPC is to be issued, the address of an RPC-accepting gateway is determined from the definition file and an RPC message is sent.
3. The RPC message is interpreted and the requested service is executed.
4. For a synchronous-response RPC, a response message from the server is received.

Example of the definition file `tran1.ini`:

```
dcrapdirect=Y
dcwatchtim=180
dcrapautoconnect=Y
dchostselect=Y
#Defines the address of the server that can process the "TRAN1" transaction.
dchost=xxx.xxx.xxx.xxx:10020,zzz.zzz.zzz.zzz:10022
```

Example of the definition file `tran2.ini`:

```
dcrapdirect=Y
dcwatchtim=180
dcrapautoconnect=Y
dchostselect=Y
#Defines the address of the server that can process the "TRAN2" transaction.
dchost=yyy.yyy.yyy.yyy:10021,zzz.zzz.zzz.zzz:10022
```

A program example that uses the above TP1/Client/J environment definition follows.

Program example

```
import JP.co.Hitachi.soft.OpenTP1.*;
public class DCCM3Caller
{
  ...
  public void Function1(){
    TP1Client clt = new TP1Client();

    // RPC that calls the TRAN1 transaction
    clt.rpcOpen("tran1.ini");
    ...
    // Synchronous-response RPC
    clt.rpcCall("dummysvg", "TRAN1", ...,
TP1Client.DCNOFLAGS);
    ...
    clt.rpcClose();

    // RPC that calls the TRAN2 transaction
    // Reloads the definition for acquiring the request target server address.
    clt.rpcOpen("tran2.ini");
    ...
    // Non-response type RPC
    clt.rpcCall("dummysvg", "TRAN2", ...,
TP1Client.DCRPC_NOREPLY);
    ...
    clt.rpcClose();
  }
}
```

(6) Notes about issuing an RPC to a DCCM3 logical terminal

- Only RPCs that use the remote API facility can be used.
- Chained RPCs cannot be used.
- The transaction control facility cannot be used.
- Before sending or receiving a message containing character string data, the character code to be used inside the message must be agreed upon with the communication-target system, and character conversion must be carried out by a UAP if necessary. In Java, Unicode is used as the internal expression of character strings inside memory.
- Specification of the `dccltinquiretime` operand of the TP1/Client/J environment definition is ignored. To specify a maximum time interval from CUP to server, use DCCM3's Unattended terminal monitoring time.
- For details about notes on DCCM3 servers, see the manuals *VOS3 Data Management System XDM E2 System Description* and *VOS1 Data Communication Management System DCCM3 Description*.

2.9.2 TCP/IP communication with DCCM3

Note the following when TCP/IP is used to communicate with a DCCM3 logical terminal:

Before sending or receiving a message containing character string data, the character code to be used inside the message must be agreed upon with the communication-target system, and character conversion must be carried out by a UAP if necessary. In Java, Unicode is used as the internal expression of character strings inside memory.

2.9.3 Sending terminal identification information to a DCCM3 logical terminal

When you wish to use a permanent connection to communicate with a DCCM3 logical terminal, you can send terminal identification information to a DCCM3 logical terminal and fix the DCCM3 logical terminal that is to be allocated to the CUP.

(1) How to send and receive messages at a DCCM3 logical terminal

To use a DCCM3 logical terminal, the communication target TP1/Client/J is defined as a logical terminal that is identified by the IP address and port number of a DCCM3 logical terminal; messages are sent and received on the basis of logical terminals.

When multiple CUPs are started from the same machine, requests from all these CUPs have the same IP address. This means that if multiple CUPs issue service requests to the same DCCM3 logical terminal port, identification of the CUPs by the DCCM3 definition is not possible. Therefore, if multiple corresponding DCCM3 logical terminals are defined, the DCCM3 logical terminal to which a CUP is allocated becomes undefined. If the DCCM3 logical terminal that is to accept a service request changes, the order of server processing at DCCM3 cannot be guaranteed. This may cause problems with some applications.

(2) Notifying the terminal identification information

When a CUP establishes permanent connection with a DCCM3 logical terminal, the DCCM3 logical terminal to be allocated to the CUP can be fixed by reporting the terminal identification information to the DCCM3 logical terminal. This is called the *terminal identification information setting facility*. This facility enables you to always allocate a CUP to the same DCCM3 logical terminal. DCCM3 calls this the *function for allocating a fixed terminal*.

The following figures show the relationship between a CUP and a DCCM3 logical terminal when the terminal identification information setting facility is not used and when it is used.

Figure 2-31: Relationship between CUP and DCCM3 logical terminal (when the terminal identification information setting facility is not used)

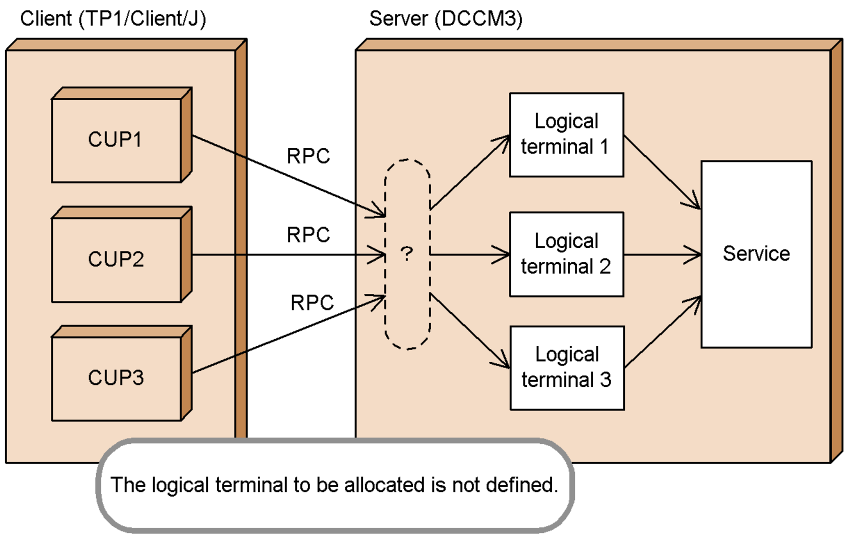
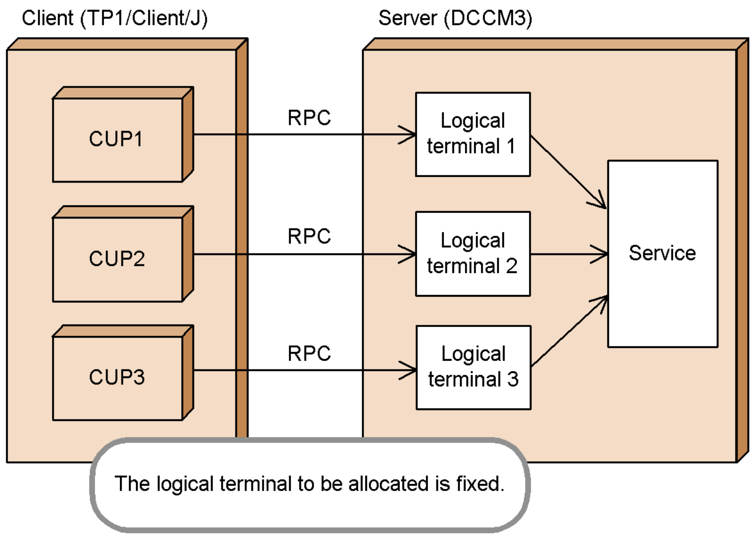


Figure 2-32: Relationship between CUP and DCCM3 logical terminal (when the terminal identification information setting facility is used)



You can use the terminal identification information setting facility by one of the following methods:

Method 1

1. Specify DCCM3 logical terminal host name in the `dchost` operand of the TP1/Client/J environment definition.
2. Specify the port number of the DCCM3 logical terminal in the `dchost` or `dcrapport` operand of the TP1/Client/J environment definition.
3. Set the terminal identification information in the `setConnectInformation` method, and then call the method.
4. Establish a permanent connection with the DCCM3 logical terminal using one of the following methods:

   - Call the `openConnection` method. If the method has parameters, specify DCCM3 logical terminal host name in the `host` parameter and the port number of the DCCM3 logical terminal in the `port` parameter.

   - Specify `Y` in the `dcrapautoconnect` operand of the TP1/Client/J environment definition, and then call the `rpcCall` method.

Method 2

1. Specify DCCM3 logical terminal host name in the `dchost` operand of the TP1/Client/J environment definition.

2. Specify the port number of the DCCM3 logical terminal in the `dchost` or `dcrapport` operand of the TP1/Client/J environment definition.
3. Set the terminal identification information in the `dccltconnectinf` operand of the TP1/Client/J environment definition.
4. Establish a permanent connection with the DCCM3 logical terminal using one of the following methods:
   - Call the `openConnection` method. If the method has parameters, specify DCCM3 logical terminal host name in the `host` parameter and the port number of the DCCM3 logical terminal in the `port` parameter.
   - Specify `Y` in the `dcrapautoconnect` operand of the TP1/Client/J environment definition and then call the `rpcCall` method.

Note:

If you set the terminal identification information in the `dccltconnectinf` operand of the TP1/Client/J environment definition and also in the `setConnectInformation` method, the setting in the `setConnectInformation` method takes effect, and the value set in the `dccltconnectinf` operand is ignored until the `rpcOpen` method is called again after the `setConnectInformation` is called.

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

- If the logical terminal name of a DCCM3 using the function for allocating a fixed terminal does not match the terminal identification information defined in TP1/Client/J, and the following methods are called, an `ErrNetDownAtClientException` exception is returned:
  - `openConnection` method
  - `rpcCall` method (applicable when `Y` is specified in the `dcrapautoconnect` operand in the TP1/Client/J environment definition)
- If terminal identification information is not set in TP1/Client/J for the logical terminal DCCM3 that uses the function for allocating a fixed terminal, but the following methods are called, an `ErrNetDownAtClientException` exception is returned:
  - `openConnection` method
  - `rpcCall` method (applicable when `Y` is specified in the `dcrapautoconnect` operand in the TP1/Client/J environment definition)
- If terminal identification information is set and establishment of a permanent connection is requested from TP1/Client/J for the logical terminal of a DCCM3

that does not use the function for allocating a fixed terminal, DCCM3 ignores the terminal identification information set by TP1/Client/J.

- If terminal identification information is set and a permanent connection with the RAP-processing server of TP1/Server is established from TP1/Client/J, the RAP-processing server ignores the terminal identification information set by TP1/Client/J. If TP1/Client/J issues an RPC to DCCM3 via the RAP-processing server, the terminal identification information set by TP1/Client/J is not transmitted to DCCM3.
- The terminal identification information is applicable only to RPCs that use the remote API facility. For an RPC that uses the name service or the scheduler direct facility, the terminal identification information is ignored, if set.

2.10 XA resource service facility

You must have uCosminexus TP1 Connector or Cosminexus TP1 Connector to use the XA resource service facility. Operations with TP1/Client/J alone are not guaranteed.

This section describes the facilities that are made available to TP1/Client/J when the XA resource service facility is used.

The following table shows the communication method supported by TP1/Client/J.

Table 2-5: Communication method supported by TP1/Client/J

TP1/Client/J communication method	Use of the remote API facility	Use of the scheduler direct facility	Use of the name service
Supported	Y	N	N

Legend:

Y: Supported

N: Not supported

The following table shows the connection modes supported by TP1/Client/J.

Table 2-6: Connection modes supported by TP1/Client/J

Connection mode	Supported
Non-auto connect mode	N
Auto connect mode	Y

Legend:

Y: Supported

N: Not supported

The following table shows the RPC call modes supported by TP1/Client/J.

Table 2-7: RPC call modes supported by TP1/Client/J

RPC call mode	Supported
Synchronous-response RPC	Y
Asynchronous-response RPC	Y
Chained RPC	Y

Legend:

Y: Supported

Note that you can specify in the `dchost` operand of the TP1/Client/J environment definition only one TP1/Server as a gateway. If you specify more than one, operations cannot be guaranteed.

2.11 Troubleshooting facility

TP1/Client/J provides a troubleshooting facility that enables you to collect UAP, data, error, memory, method, debug, performance analysis, and performance verification traces.

The facility outputs UAP, data, error, and method traces to files. A memory trace is stored in the String-type array provided by the CUP. A debug trace is collected in the memory of TP1/Client/J. When a method provided by TP1/Client/J returns an exception, its trace is output to a file or to the standard output. A performance analysis trace is output on the Cosminexus Application Server and a performance verification trace is output on the TP1/Server.

For Java applets, the UAP, data, error, method, and performance verification traces are not available due to Java security limitations. Furthermore, a performance analysis trace cannot be collected.

For Java servlets, some traces are not available to some application servers due to security limitations. For details about the security limitations for Java servlets, see the documentation for the applicable application server.

To output traces, you must set the time zone in the `TZ` environment variable. If this environment variable is not set, the correct time will not be output for trace information.

2.11.1 Contents of trace files

The following table lists the options that can be specified for the trace files and the output file names.

Table 2-8: Options and names for trace files

Trace file	Options	Where to specify the file destination	Output file name
UAP trace	• File output directory • Size of trace file	`TrcPath` argument of the `setUapTraceMode` method or `dcuaptracepath` operand	`DCUAP1.TRC` `DCUAP2.TRC`
Data trace	• File output directory • Size of trace file • Maximum size of data	`TrcPath` argument of the `setDataTraceMode` method or `dcdatatracepath` operand	`DCDAT1.TRC` `DCDAT2.TRC`
Error trace	• File output directory • Size of trace file	`TrcPath` argument of the `setErrorTraceMode` method or `dcerrtracepath` operand	`DCERR1.TRC` `DCERR2.TRC`

Trace file	Options	Where to specify the file destination	Output file name
Method trace	• File output directory • Size of trace file	`TrcPath` argument of the `setMethodTraceMode` method or `dcmethodtracepath` operand	`DCMTD1.TRC` `DCMTD2.TRC`

TP1/Client/J cannot output a trace file if the specified directory does not exist or if there is no write authority for the specified directory. If the specified target directory does not contain a file with the predefined output file name, TP1/Client/J creates the trace file in the applicable directory. Data is written into a file in the addition mode. TP1/Client/J uses two trace files with the round robin method. When the amount of data exceeds the specified trace file size, TP1/Client/J swaps the files and restarts the write operation from the beginning of the other file.

If you output the trace files for multiple CUPs to the same directory, the trace information for the various CUPs is intermingled, making troubleshooting difficult. For this reason, you should output the trace files for each CUP to a separate directory. Note that the actual size of a trace file may be larger than the specified value depending on the output size of one trace data item.

2.11.2 UAP trace

TP1/Client/J outputs UAP trace information to the directory specified in the `TrcPath` argument of the `setUapTraceMode` method or the directory specified in the `dcuaptracepath` operand of the TP1/Client/J environment definition; the file names are `DCUAP1.TRC` and `DCUAP2.TRC`. You specify the file size with the `size` argument of the `setUapTraceMode` method or the `dcuaptracesize` operand of the TP1/Client/J environment definition.

For a UAP trace, TP1/Client/J collects the following information at the start and end of the method provided by TP1/Client/J:

- Date
- Time
- Name of the executing thread
- Name of the called method
- Exception that occurred
- Information about specified arguments
- Information about the called method
- Contents of transferred data

If the transferred data exceeds 60 bytes, only the first 60 bytes of data are obtained.

The output format of a UAP trace file is as follows:

```
yyyy/mm/dd hh:mm:ss.uuu Location = lll ThreadName = ttt
MethodName = nnnnnnnnnnnnnnnnnnnn
Exception = eeeeeeeeeeeeeeeeeeee
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
................
00000020 73 61 69 73 76 72 30 31 00 00 00 00 00 00 00 00
saisvr01........
:         :        :
```

- *yyyy/mm/dd hh:mm:ss.uuu*: Date and time the UAP trace was collected
- *lll*: Type of entry or exit point
- *ttt*: Name of the executing thread
- *nnnnnnnnnnnnnnnnnnnn*: Name of the called method
- *eeeeeeeeeeeeeeeeeeee*: Exception that occurred
- Character string following ADDRESS: Various items of information (information about the specified arguments, information about the called method, and the contents of transferred data)

The information that is output depends on the method and collection timing. The following tables show the information that is output for each method.

Table 2-9: Information for the openConnection() method and the openConnection(host, port) method

Collection location	Item	Location		Length (bytes)	Description
		Hexadecimal	Decimal		
Entry point	Connection-target host name	0	0	64	Connection-target host name specified in the argument. If the argument is not specified, this information is not collected, in which case this field is cleared to zeros.

Collection location	Item	Location		Length (bytes)	Description
		Hexadecimal	Decimal		
	Connection-target port number	40	64	4	Connection-target port number specified in the argument. If the argument is not specified, this information is not collected, in which case a UAP trace is obtained with `-1` set.
	Maximum time of inquiry interval	44	68	4	Maximum time of inquiry interval used during method execution. This is the value specified in the `dccltinquiretime` operand of the TP1/Client/J environment definition or in the `setDccltinquiretime` method.
	Maximum time to wait for a response	48	72	4	Maximum time to wait for a response during method execution. This is the value specified in the `dcwatchtim` operand of the TP1/Client/J environment definition or in the `setDcwatchtim` method.
	Input stream checking interval	4C	76	4	Input stream checking interval used during method execution. This is the value specified in the `dcselint` operand of the TP1/Client/J environment definition or in the `setDcselint` method.
Exit point	Performance analysis trace identification information	0	0	26	Performance analysis trace identification information

Table 2-10: Information for the openConnection(url, flags) method

Collection location	Item	Location		Length (bytes)	Description
		Hexadecimal	Decimal		
Entry point	Connection destination URL	0	0	128	TP1/Web URL specified by the `url` parameter.
	Option flags	0	0	2	Value specified by the `openConnection` `flags` parameter. The following value is set: • `DCSESSION 0x0001`
	Maximum time to wait for a response	0	0	4	The maximum time to wait for a response specified by the `setDcwatchtim` method.
Exit point	None	N/A	N/A	N/A	N/A

N/A: Not applicable

Table 2-11: Information for the rpcCall method

Collection location	Item	Location		Length (bytes)	Description
		Hexadecimal	Decimal		
Entry point	Service group name	0	0	32	Service group name of the called service
	Service name	20	32	32	Name of the called service
	Send data length	40	64	4	Length of send data
	Send data	44	68	60	Data to be sent
	Receive data length	80	128	4	Length of data to be received

Collection location	Item	Location		Length (bytes)	Description
		Hexadecimal	Decimal		
	Option flag	84	132	4	Value specified for the `flags` argument of the `rpcCall` method. One of the following values is set: • `DCNOFLAGS` `0x00000000` • `DCRPC_NOREPLY` `0x00000001` • `DCRPC_CHAINED` `0x00000004` • `DCRPC_TPNOTRAN` `0x00000020`
	Maximum time to wait for a response	88	136	4	Maximum time to wait for a response during method execution. This is the value specified in the `dcwatchtim` operand of the TP1/Client/J environment definition or in the `setDcwatchtim` method.
	Server communication delay time	8C	140	4	Server communication delay time used during method execution. This is the value specified in the `dccltdelay` operand of the TP1/Client/J environment definition or in the `setDccltdelay` method. `0` is set if the remote API facility is not used.
	Input stream checking interval	90	144	4	Input stream checking interval used during method execution. This is the value specified in the `dcselint` operand of the TP1/Client/J environment definition or in the `setDcselint` method.

Collection location	Item	Location		Length (bytes)	Description
		Hexadecimal	Decimal		
Exit point#	Receive data length	0	0	4	Length of the received response
	Receive data	4	4	60	Received data
	Performance analysis trace identification information	40	64	26	Performance analysis trace identification information

#: In the case of a no-response-type service request, the exit point information consists of performance analysis trace identification information only.

Table 2-12: Information for the rpcCallTo method

Collection location	Item	Location		Length (bytes)	Description
		Hexadecimal	Decimal		
Entry point	Connection-target host name	0	0	64	Target host name used during method execution. This is the value specified in `DCRpcBindTbl`.
	Connection-target port number	40	64	4	Port number of the target host that was used during method execution. This is the value specified in `DCRpcBindTbl`.
	Connection-target option flag	44	68	4	Communication protocol flag of the `rpcCallTo` method: • `DCRPC_SCDPORT` `0x00000002`
	Service group name	48	72	32	Service group name of the called service
	Service name	68	104	32	Name of the called service
	Send data length	88	136	4	Length of send data
	Send data	8C	140	60	Data to be sent
	Receive data length	C8	200	4	Length of data to be received

Collection location	Item	Location		Length (bytes)	Description
		Hexadecimal	Decimal		
	Option flag	CC	204	4	Value specified in the `flags` argument of the `rpcCallTo` method. One of the following values is set: • `DCNOFLAGS` `0x00000000` • `DCRPC_NOREPLY` `0x00000001`
	Maximum time to wait for a response	D0	208	4	Maximum time to wait for a response during method execution. This is the value specified in the `dcwatchtim` operand of the TP1/Client/J environment definition or in the `setDcwatchtim` method.
	Server communication delay time	D4	212	4	Server communication delay time used during method execution. This is the value specified in the `dccltdelay` operand of the TP1/Client/J environment definition or in the `setDccltdelay` method. `0` is set if the remote API facility is not used.
	Input stream checking interval	D8	216	4	Input stream checking interval used during method execution. This is the value specified in the `dcselint` operand of the TP1/Client/J environment definition or in the `setDcselint` method.
Exit point#	Receive data length	0	0	4	Length of the received response
	Receive data	4	4	60	Received data
	Performance analysis trace identification information	40	64	26	Performance analysis trace identification information

#: In the case of a no-response-type service request, the exit point information consists of performance analysis trace identification information only.

Table 2-13: Information for the closeConnection method

Collection location	**Item**	**Location**		**Length (bytes)**	**Description**
		Hexadecimal	**Decimal**		
Entry point	Maximum time to wait for a response	0	0	4	Maximum time to wait for a response during method execution. This is the value specified in the `dcwatchtim` operand of the TP1/Client/J environment definition or in the `setDcwatchtim` method.
	Input stream checking interval	4	4	4	Input stream checking interval used during method execution. This is the value specified in the `dcselint` operand of the TP1/Client/J environment definition or in the `setDcselint` method.
Exit point	Performance analysis trace identification information	0	0	26	Performance analysis trace identification information

Table 2-14: Information for the setDcwatchtim method

Collection location	**Item**	**Location**		**Length (bytes)**	**Description**
		Hexadecimal	**Decimal**		
Entry point	Maximum time to wait for a response	0	0	4	Maximum time to wait for a response
Exit point	None	N/A	N/A	N/A	N/A

N/A: Not applicable

Table 2-15: Information for the setDccltinquiretime method

Collection location	Item	Location		Length (bytes)	Description
		Hexadecimal	Decimal		
Entry point	Maximum time of inquiry interval	0	0	4	Maximum value of inquiry interval
Exit point	None	N/A	N/A	N/A	N/A

N/A: Not applicable

Table 2-16: Information for the setDccltdelay method

Collectio n location	Item	Location		Length (bytes)	Description
		Hexadecimal	Decimal		
Entry point	Server communication delay time	0	0	4	Server communication delay time
Exit point	None	N/A	N/A	N/A	N/A

N/A: Not applicable

Table 2-17: Information for the setDcselint method

Collection location	Item	Location		Length (bytes)	Description
		Hexadecimal	Decimal		
Entry point	Input stream checking interval	0	0	4	Input stream checking interval
Exit point	None	N/A	N/A	N/A	N/A

N/A: Not applicable

Table 2-18: Information for the setDccltextend method

Collection location	Item	Location		Length (bytes)	Description
		Hexadecimal	Decimal		
Entry point	Level of extended facility	0	0	4	Extension level of the TP1/Client/J facility
Exit point	None	N/A	N/A	N/A	N/A

N/A: Not applicable

Table 2-19: Information for the setRpcextend method

Collection location	Item	Location		Length (bytes)	Description
		Hexadecimal	Decimal		
Entry point	Level of RPC extended facility	0	0	4	Extension level of the RPC facility of TP1/Client/J. Disjunction of the following values is set: • `DCRPC_SCD_LOAD_PRIORITY 0x00000008` • `DCRPC_WATCHTIMINHERIT 0x00000010` • `DCRPC_RAP_AUTOCONNECT 0x00000020` • `DCRPC_WATCHTIMRPCINHERIT 0x00000040`
Exit point	None	N/A	N/A	N/A	N/A

N/A: Not applicable

Table 2-20: Information for the setDchost method

Collection location	Item	Location		Length (bytes)	Description
		Hexadecimal	Decimal		
Entry point	Connection-target host name	0	0	64	Host name specified in the `host` argument
	Connection-target port number	40	64	4	Port number specified in the `port` argument
Exit point	None	N/A	N/A	N/A	N/A

N/A: Not applicable

Table 2-21: Information for the rpcOpen method

Collection location	Item	Location		Length (bytes)	Description
		Hexadecimal	Decimal		
Entry point	Name of the TP1/ Client/J environment definition file	0	0	256	TP1/Client/J environment definition file name specified in the `deffilename` argument. If the argument is not specified, this information is not collected, in which case this field is cleared to zeros.
Exit point	None	N/A	N/A	N/A	N/A

N/A: Not applicable

Table 2-22: Information for the trnBegin method

Collection location	Item	Location		Length (bytes)	Description
		Hexadecimal	Decimal		
Entry point	Expiry time in transaction branch	0	0	4	Value specified in the `dccltrexptm` operand of the TP1/Client/J environment definition
	Transaction branch CPU monitoring interval	4	4	4	Value specified in the `dcclttrcputm` operand of the TP1/Client/J environment definition
	Object of transaction branch monitoring interval	8	8	1	Value specified in the `dcclttrexpsp` operand of the TP1/Client/J environment definition: • `Y 89` • `N 78` • `F 70`

Collection location	Item	Location		Length (bytes)	Description
		Hexadecimal	Decimal		
	Statistical information collection item	9	9	4	Value specified in the `dccltstatisitem` operand of the TP1/Client/J environment definition: • `base 0x80000000` • `executiontime 0x40000000` • `cputime 0x20000000` • `function 0x10000000`
	Transaction optimization item	D	13	4	Value specified in the `dccltroptiitem` operand of the TP1/Client/J environment definition: • `nothing 0` • `base 0x00000003` • `asyncprepare 0x00000004` • `recursivemigrate 0x00000008`
	Maximum communication wait time during transaction synchronization point processing	11	17	4	Value specified in the `dccltrwatchtime` operand of the TP1/Client/J environment definition
	Value for rollback information acquisition	15	21	4	Value specified in the `dcclttrrbinfo` operand of the TP1/Client/J environment definition: • `no 0` • `self 0x00000001` • `remote 0x00000002` • `all 0x00000003`
	Maximum executable value for transaction branch	19	25	4	Value specified in the `dccltrlimittime` operand of the TP1/Client/J environment definition

Collection location	Item	Location		Length (bytes)	Description
		Hexadecimal	Decimal		
	Value for rollback completion report	1D	29	4	Value specified in the `dcclttrrbrcv` operand of the TP1/Client/J environment definition: • `Y 89` • `N 78`
	Value for synchronization point processing method in the event of a UAP error	21	33	4	Value specified in the `dccltrrecoverytype` operand of the TP1/Client/J environment definition: • `type1 0x00000001` • `type2 0x00000002` • `type3 0x00000004`
Exit point	Performance analysis trace identification information	0	0	26	Performance analysis trace identification information

Table 2-23: Information for the trnInfo method

Collection location	Item	Location		Length (bytes)	Description
		Hexadecimal	Decimal		
Entry point	None	N/A	N/A	N/A	N/A
Exit point	Under-transaction flag	0	0	1	• `1`: Under transaction • `0`: Not under transaction

N/A: Not applicable

Table 2-24: Information for the getTrnID method

Collection location	Item	Location		Length (bytes)	Description
		Hexadecimal	Decimal		
Entry point	None	N/A	N/A	N/A	N/A
Exit point	Transaction global identifier	0	0	16	This information is acquired only if the method terminates normally.
	Transaction branch identifier	10	16	16	This information is acquired only if the method terminates normally.

N/A: Not applicable

Table 2-25: Information for the cltReceive method

Collection location	Item	Location		Length (bytes)	Description
		Hexadecimal	Decimal		
Entry point	Receive data length	0	0	4	Length of data to be received
	Timeout set time	4	4	4	Time interval specified for timeout
	Option flags	8	8	4	Value specified by the `flags` parameter of the `cltReceive` method. One of the following values is set: • `DCNOFLAGS` `0x00000000` • `DCCLT_RCV_CLOSE` `0x00000002`
Exit point	Receive data length	0	0	4	Length of data to be received
	Receive data	4	4	60	Received data

Table 2-26: Information for the cltSend method

Collection location	Item	Location		Length (bytes)	Description
		Hexadecimal	Decimal		
Entry point	Send data	0	0	60	Data to be sent
	Send data length	3C	60	4	Length of data to be sent
	Name of connected host	40	64	64	Name of remote host specified by the `hostname` parameter.
	Connection destination port number	80	128	4	Port number of remote system specified by the `portnum` parameter.
	Option flags	84	132	4	Value specified by the `flags` parameter of the `cltSend` method. One of the following values is set: • `DCNOFLAGS 0x00000000` • `DCCLT_SND_CLOSE 0x00000001`
Exit point	None	N/A	N/A	N/A	N/A

N/A: Not applicable

Table 2-27: Information for the cltAssemSend method

Collection location	Item	Location		Length (bytes)	Description
		Hexadecimal	Decimal		
Entry point	Send data	0	0	60	Data to be sent
	Send data length	3C	60	4	Length of data to be sent
	Name of connected host	40	64	64	Name of remote host specified by the `hostname` parameter.
	Connection destination port number	80	128	4	Port number of remote system specified by the `portnum` parameter.
	Time-out setting	84	132	4	Time-out setting

Collection location	Item	Location		Length (bytes)	Description
		Hexadecimal	Decimal		
	Option flags	88	136	4	Value specified by the `flags` parameter of the `cltAssemSend` method. One of the following values is set: • `DCNOFLAGS 0x00000000` • `DCCLT_SND_CLOSE 0x00000001`
Exit point	None	N/A	N/A	N/A	N/A

N/A: Not applicable

Table 2-28: Information for the cltAssemReceive method

Collection location	Item	Location		Length (bytes)	Description
		Hexadecimal	Decimal		
Entry point	Time-out setting	0	0	4	Time-out setting
	Option flags	4	4	4	Value specified by the `flags` parameter of the `cltAssemReceive` method. One of the following values is set: • `DCNOFLAGS 0x00000000` • `DCCLT_RCV_CLOSE 0x00000002`
Exit point	Receive data length	0	0	4	Length of response received
	Receive data	4	4	60	Received data

Table 2-29: Information for the setConnectInformation method

Collection location	Item	Location		Length (bytes)	Description
		Hexadecimal	Decimal		
Entry point	Logical terminal name of the DCCM3 logical terminal	0	0	64	Logical terminal name of the DCCM3 logical terminal, as specified in the `inf` parameter

Collection location	Item	Location		Length (bytes)	Description
		Hexadecimal	Decimal		
	Length of terminal identification information	40	64	2	Length of terminal identification information, as specified in the `inf_len` parameter
Exit point	None	N/A	N/A	N/A	N/A

N/A: Not applicable

Table 2-30: Information for the acceptNotification method

Collection location	Item	Location		Length (bytes)	Description
		Hexadecimal	Decimal		
Entry point	Length of area for storing a notification message from the server	0	0	4	Length of area for storing a notification message from the server, as specified in the `inf_len` parameter
	Port number for receiving a notification message from the server	4	4	4	Port number for receiving a notification message from the server, as specified in the `port` parameter
	Timeout value	8	8	4	Timeout value specified in the `timeout` parameter
Exit point	Notification message from the server	0	0	60	Notification message from the server that was stored in the `inf` parameter
	Length of notification message from the server	3C	60	4	Length of notification message from the server, as specified in the `inf_len` argument
	Host name of the server	40	64	64	Server's host name stored in the `hostname` parameter
	Node identifier of the server	80	128	8	Server's node identifier specified in the `nodeid` parameter

Table 2-31: Information for the cancelNotification method

Collection location	Item	Location		Length (bytes)	Description
		Hexadecimal	Decimal		
Entry point	Message to be sent to CUP	0	0	60	Message to be sent to the CUP, as specified in the `inf` parameter
	Length of message to be sent to CUP	3C	60	4	Length of the message to be sent to CUP, as specified in the `inf_len` parameter
	Host name of CUP waiting to receive unidirectional message	40	64	64	Host name of the CUP waiting to receive the unidirectional message, as specified in the `hostname` parameter
	Port number of CUP waiting to receive unidirectional message	80	128	4	Port number of the CUP waiting to receive the unidirectional message, as specified in the `port` parameter
Exit point	None	N/A	N/A	N/A	N/A

N/A: Not applicable

Table 2-32: Information for the openNotification method

Collection location	Item	Location		Length (bytes)	Description
		Hexadecimal	Decimal		
Entry point	Port number for receiving a notification message from the server	0	0	4	Port number for receiving a notification message from the server, as specified in the `port` parameter
Exit point	None	N/A	N/A	N/A	N/A

N/A: Not applicable

Table 2-33: Information for the acceptNotificationChained method

Collection location	Item	Location		Length (bytes)	Description
		Hexadecimal	Decimal		
Entry point	Length of area for storing a notification message from the server	0	0	4	Length of area for storing a notification message from the server, as specified in the `inf_len` parameter
	Timeout value	4	4	4	Timeout value specified in the `timeout` parameter
Exit point	Notification message from the server	0	0	60	Notification message from the server, as stored in the `inf` parameter
	Length of notification message from the server	3C	60	4	Length of the notification message from the server, as specified in the `inf_len` argument
	Host name of the server	40	64	64	Server's host name stored in the `hostname` argument
	Node identifier of the server	80	128	8	Server's node identifier specified in the `nodeid` parameter

Table 2-34: Information for the setUapTraceMode method

Collection location	Item	Location		Length (bytes)	Description
		Hexadecimal	Decimal		
Entry point	UAP trace output directory	0	0	256	UAP trace output directory specified in the `TrcPath` parameter
	Size of UAP trace file to be output	100	256	4	Size of the output UAP trace file, as specified in the `size` parameter
	Flag indicating whether or not UAP trace is to be acquired	104	260	4	Flag indicating whether or not UAP trace is to be acquired, as specified in the `flag` parameter (if the `flag` parameter is set to `true`, `1` is output; if it is set to `false`, `0` is output)

Collection location	Item	Location		Length (bytes)	Description
		Hexadecimal	Decimal		
Exit point	None	N/A	N/A	N/A	N/A

N/A: Not applicable

Table 2-35: Information for the setErrorTraceMode method

Collection location	Item	Location		Length (bytes)	Description
		Hexadecimal	Decimal		
Entry point	Error trace output directory	0	0	256	Error trace output directory specified in the `TrcPath` parameter
	Size of error trace file to be output	100	256	4	Size of the output error trace file, as specified in the `size` parameter
	Flag indicating whether or not error trace is to be acquired	104	260	4	Flag indicating whether or not error trace is to be acquired, as specified in the `flag` parameter (if the `flag` parameter is set to `true`, `1` is output; if it is set to `false`, `0` is output)
Exit point	None	N/A	N/A	N/A	N/A

N/A: Not applicable

Table 2-36: Information for the setMethodTraceMode method

Collection location	Item	Location		Length (bytes)	Description
		Hexadecimal	Decimal		
Entry point	Method trace output directory	0	0	256	Method trace output directory specified in the `TrcPath` parameter
	Size of method trace file to be output	100	256	4	Size of the output method trace file, as specified in the `size` parameter

Collection location	Item	Location		Length (bytes)	Description
		Hexadecimal	Decimal		
	Flag indicating whether or not method trace is to be acquired	104	260	4	Flag indicating whether or not method trace is to be acquired, as specified in the `flag` parameter (if the `flag` parameter is set to `true`, `1` is output; if it is set to `false`, `0` is output)
Exit point	None	N/A	N/A	N/A	N/A

N/A: Not applicable

Table 2-37: Information for the setDataTraceMode method

Collection location	Item	Location		Length (bytes)	Description
		Hexadecimal	Decimal		
Entry point	Data trace output directory	0	0	256	Data trace output directory specified in the `TrcPath` parameter
	Size of data trace file to be output	100	256	4	Size of the output data trace file, as specified in the `size` parameter
	Size of data to be output to the data trace	104	260	4	Size of the data to be output to the data trace, as specified in the `DataSize` parameter
	Flag indicating whether or not data trace is to be acquired	108	264	4	Flag indicating whether or not data trace is to be acquired, as specified in the `flag` parameter (if the `flag` parameter is set to `true`, `1` is output; if it is set to `false`, `0` is output)
Exit point	None	N/A	N/A	N/A	N/A

N/A: Not applicable

Table 2-38: Information for the trnChainedCommit method

Collection location	Item	Location		Length (bytes)	Description
		Hexadecimal	Decimal		
Entry point	None	N/A	N/A	N/A	N/A
Exit point	Performance analysis trace identification information	0	0	26	Performance analysis trace identification information

N/A: Not applicable

Table 2-39: Information for the trnUnchainedCommit method

Collection location	Item	Location		Length (bytes)	Description
		Hexadecimal	Decimal		
Entry point	None	N/A	N/A	N/A	N/A
Exit point	Performance analysis trace identification information	0	0	26	Performance analysis trace identification information

N/A: Not applicable

Table 2-40: Information for the trnChainedRollback method

Collection location	Item	Location		Length (bytes)	Description
		Hexadecimal	Decimal		
Entry point	None	N/A	N/A	N/A	N/A
Exit point	Performance analysis trace identification information	0	0	26	Performance analysis trace identification information

N/A: Not applicable

Table 2-41: Information for the trnUnchainedRollback method

Collection location	Item	Location		Length (bytes)	Description
		Hexadecimal	Decimal		
Entry point	None	N/A	N/A	N/A	N/A

Collection location	Item	Location		Length (bytes)	Description
		Hexadecimal	Decimal		
Exit point	Performance analysis trace identification information	0	0	26	Performance analysis trace identification information

N/A: Not applicable

2.11.3 Data trace

TP1/Client/J outputs data trace information to the directory specified in the `TrcPath` argument of the `setDataTraceMode` method or the directory specified in the `dcdatatracepath` operand of the TP1/Client/J environment definition; the file names are `DCDAT1.TRC` and `DCDAT2.TRC`. You specify the file size with the `size` argument of the `setDataTraceMode` method or the `dcdatatracesize` operand of the TP1/Client/J environment definition. To specify the data size of trace 1, use the `DataSize` argument of the `setDataTraceMode` method or the `dcdatatracemaxsize` operand of the TP1/Client/J environment definition.

For a data trace, TP1/Client/J collects descriptions of the messages transferred between the CUP and the TP1/Server.

The output format of a data trace file is as follows:

```
yyyy/mm/dd hh:mm:ss.uuu Event = eeee ThreadName = ttt
ADDRESS +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +a +b +c +d +e +f
0123456789abcdef
00000000 07 70 c0 00 00 00 00 5c 00 00 00 00 00 06 00 05
.p.....\........
00000010 00 00 00 10 00 00 00 00 00 00 00 00 15 bf 00 00
................
00000020 00 00 00 00 00 00 00 00 00 00 00 00 00 01 01 00
................
:         :        :
```

- *yyyy*/*mm*/*dd* *hh*:*mm*:*ss*.*uuu*: Date and time the data trace was collected
- *eeee*: Send/receive type
- *ttt*: Name of the executing thread
- Character string following `ADDRESS`: Data

2.11.4 Error trace and memory trace

TP1/Client/J outputs error trace information to the directory specified in the `TrcPath` argument of the `setErrorTraceMode` method or the directory specified in the `dcerrtracepath` operand of the TP1/Client/J environment definition; the file names are `DCERR1.TRC` and `DCERR2.TRC`. You specify the file size with the `size` argument of the `setErrorTraceMode` method or the `dcerrtracesize` operand of the TP1/Client/J environment definition.

For an error trace, the errors detected by TP1/Client/J are recorded in the files in message format. If an error occurs during method execution, its cause is reported as an exception, but this information may not be sufficient to identify the cause. An error trace provides detailed error information in the file; you can use this information to determine the cause of an error.

A memory trace is a facility for storing error information in the `String` array specified in the `setTraceArray` method when you use a Java applet for which an error trace cannot be collected. In the event of an error, you can obtain error information by referencing the `String` array.

The output format of an error trace file or the format of data stored in a `String` array for a memory trace is as follows:

(*ttt*)*yyyy*/*mm*/*dd* *hh*:*mm*:*ss*.*uuu* *eeeeeeeeeeee*

- *ttt*: Name of the executing thread
- *yyyy*/*mm*/*dd* *hh*:*mm*:*ss*.*uuu*: Date and time the error trace or memory trace was collected
- *eeeeeeeeeeee*: Message

The following table lists the messages that are collected.

Table 2-42: Messages collected in error and memory traces

Message	Description
`Invalid message received.` `method=`*aaaaaaaa*	An invalid message was received from the TP1/Server. *aaaaaaaa*: Name of the method that issued this message. For a memory trace, this is a code corresponding to the name of the method that issued this message.

`Error reply received.`
`inf=`*aaaaaaaa*`,`
`method=`*bbbbbbbb*
	An error reply was received from the TP1/Server.
	aaaaaaaa: Received error code
	bbbbbbbb: Name of the method that issued this message.
	For a memory trace, this is a code corresponding to the name of the method that issued this message.

Message	Description

`Exception occurred.`
`inf=`*aaaaaaaa*`,`
`exception=`*bbbbbbbb*`(`*cc...cc*`),`
`method=`*dddddddd*
	An exception was received from the Java system in the TP1Client class. Or, an exception was returned to the Java applet, application, or servlet from the TP1Client class.
	aaaaaaaa: Maintenance information when the exception occurred.
	bbbbbbbb: Name of the exception that was received from Java or that was returned to the Java applet, application, or servlet.
	For a memory trace, this is a code corresponding to the name of the exception.
	cc...cc: Detailed message for the exception.
	This information is output only when there is a detailed message.
	dddddddd: Name of the method that issued this message.
	For a memory trace, this is a code corresponding to the method name.

`Invalid data received.` `(`*aa...aa*`), method=`*bbbbbbbb*	The `cltAssemReceive` method received invalid data from the remote system. *aa...aa*: Invalid data When the message length is invalid `receive message length=`*message-length* (decimal) *bbbbbbbb*: Name of the method that issued this message. For a memory trace, this is a code corresponding to the name of the method that issued this message.

`Receiving message was canceled.`
aaaaaaaa `(`*bb...bb*`) method=`*cccccccc*
	The message received by the `cltAssemReceive` method from the remote system was discarded.
	aaaaaaaa: Reason for discarding the message
	bb...bb: Data detail
	When the receive buffer overflowed
	`receive buffer overflowed.`
	`(receive buffer size=`*receive-buffer-size* (decimal),
	`receive message body`
	`length=`*receive-message-body-length* (decimal)
	cccccccc: Name of the method that issued this message.
	For a memory trace, this is a code corresponding to the name of the method that issued this message.

Message	Description
`User data did not compress,` `group=`*aa...aa*`,` `service=`*bb...bb*`,` `reason=` *cc...cc*	User data was not compressed. The service request is issued without compressing the user data. *aa...aa*: Target service group name *bb...bb*: Target service name *cc...cc*: Reason why the user data was not compressed: `NO EFFECT`: Compressing the user data would have no effect. `NOT SUPPORT VERSION`: The target TP1/Server version does not support the data compression facility. If `NO EFFECT` is displayed, the size of the data obtained after compression exceeds the size of the uncompressed data. In such a case, check to see if this message has been output more than once for the same CUP, and re-evaluate the use of the data compression facility for each CUP. If `NOT SUPPORT VERSION` is displayed, check whether the target TP1/Server version supports the data compression facility (TP1/Server Base 03-03 or later).

The following table lists the method names that correspond to the codes that are output to a memory trace.

Table 2-43: Correspondence of codes to method names

Code	Method name
1	`TP1Client.openConnection`
2	`TP1Client.closeConnection`
3	`TP1Client.rpcCall`
4	`TP1Client.setDccltinquiretime`
5	`TP1Client.setDccltdelay`
6	`TP1Client.setDcwatchtim`
7	`TP1Client.setDcselint`
8	`TP1Client.setDccltextend`
9	`TP1Client.rpcOpen`
10	`TP1Client.rpcClose`
11	`TP1Client.setRpcextend`
12	`TP1Client.setDchost`
13	`TP1Client.rpcCallTo`

Code	Method name
14	TP1Client.trnBegin
15	TP1Client.trnChainedCommit
16	TP1Client.trnChainedRollback
17	TP1Client.trnUnchainedCommit
18	TP1Client.trnUnchainedRollback
19	TP1Client.trnInfo
20	TP1Client.getTrnID
37	TP1Client.cltAssemSend
38	TP1Client.cltAssemReceive
100	TP1ClientSocketCommunicator.openConnection
101	TP1ClientSocketCommunicator.closeConnection
102	TP1ClientSocketCommunicator.sendData
103	TP1ClientSocketCommunicator.sendData
104	TP1ClientSocketCommunicator.recvData
105	TP1ClientSocketCommunicator.recvData
106	TP1ClientSocketCommunicator.recvDummyData
107	TP1ClientSocketCommunicator.flush
108	TP1ClientSocketCommunicator.recvSelect
109	TP1ClientSocketCommunicator.openServerSocket
110	TP1ClientSocketCommunicator.acceptServerSocket
111	TP1ClientSocketCommunicator.closeServerSocket
112	TP1ClientSocketCommunicator.getServerPort
113	TP1ClientSocketCommunicator.getLocalIPAddress
114	TP1ClientSocketCommunicator.getLocalPort
300	TP1ClientProperties.TP1ClientProperties
301	TP1ClientProperties.TP1ClientProperties
302	TP1ClientProperties.getValue

Code	Method name
400	`TP1ClientRpc.rpcOpen`
401	`TP1ClientRpc.cltConnect`
402	`TP1ClientRpc.rpcCall`
403	`TP1ClientRpc.rpcClose`
404	`TP1ClientRpc.cltDisconnect`
405	`TP1ClientRpc.defAnalyze`
406	`TP1ClientRpc.rapConnect`
407	`TP1ClientRpc.rapDisconnect`
408	`TP1ClientRpc.rapRpcCall`
409	`TP1ClientRpc.scdRpcCall`
410	`TP1ClientRpc.setDccltextend`
411	`TP1ClientRpc.rapMngConnect`
412	`TP1ClientRpc.rapMngDisconnect`
413	`TP1ClientRpc.setRpcextend`
414	`TP1ClientRpc.setDchost`
415	`TP1ClientRpc.namRpcCall`
416	`TP1ClientRpc.getHostEntry`
417	`TP1ClientRpc.getNextEntry`
500	`TP1ClientConManage.openMngConnection`
501	`TP1ClientConManage.closeMngConnection`
502	`TP1ClientConManage.changeMngConnection`
503	`TP1ClientConManage.getConnection`
504	`TP1ClientConManage.putConnection`
505	`TP1ClientConManage.cancelConnection`
506	`TP1ClientConManage.registCheck`
507	`TP1ClientConManage.getMngConInfo`
512	`TP1ClientConManage.getSync`

Code	Method name
513	TP1ClientConManage.registConnNum
600	TP1ClientConnectionHost.addTP1ClientConnectionHost
601	TP1ClientConnectionHost.removeTP1ClientConnectionHost
602	TP1ClientConnectionHost.removeTP1ClientConnectionHostAll
603	TP1ClientConnectionHost.changeTP1ClientConnectionHost
604	TP1ClientConnectionHost.getConnection
605	TP1ClientConnectionHost.putConnection
606	TP1ClientConnectionHost.addConnection
700	TP1ClientNam.Lookup
900	TP1ClientTrn.trnBegin
901	TP1ClientTrn.trnChainedCommit
902	TP1ClientTrn.trnChainedRollback
903	TP1ClientTrn.trnUnchainedCommit
904	TP1ClientTrn.trnUnchainedRollback
905	TP1ClientTrn.conTrnCall
1001	Socket.Socket
1002	Socket.getInputStream
1003	Socket.getOutputStream
1004	Socket.close
1005	DataInputStream.read
1006	DataInputStream.available
1007	DataInputStream.close
1008	DataOutputStream.write
1009	DataOutputStream.close
1010	InetAddress.getLocalHost
1013	InputStream.read
1014	InputStream.close

Code	Method name
1015	`OutputStream.write`
1016	`OutputStream.close`
1019	`Socket.setTcpNoDelay`
1102	`TP1ClientSndRcv.cltAssemSend`
1103	`TP1ClientSndRcv.cltAssemReceive`

The following table lists the exception names that correspond to the codes that are output to a memory trace.

Table 2-44: Correspondence of codes to exception names

Code	Exception name
1	`ErrInvalidArgsException`
2	`ErrProtoException`
3	`ErrNoBufsException`
4	`ErrNetDownException`
5	`ErrTimedOutException`
6	`ErrMessageTooBigException`
7	`ErrReplyTooBigException`
8	`ErrNoSuchServiceGroupException`
9	`ErrNoSuchServiceException`
10	`ErrServiceClosedException`
11	`ErrServiceTerminatingException`
12	`ErrServiceNotUpException`
13	`ErrNotUpException`
14	`ErrSyserrAtServerException`
15	`ErrNoBufsAtServerException`
16	`ErrSyserrException`
17	`ErrInvalidReplyException`
18	`ErrInitializingException`

Code	Exception name
19	ErrServerBusyException
20	ErrTestmodeException
21	ErrSecchkException
22	ErrServiceTerminatedException
23	ErrIOErrException
24	ErrHostUndefException
25	ErrInvalidPortException
26	ErrConnfreeException
29	ErrFatalException
30	ErrSecurityException
31	NumberFormatException
32	EOFException
33	FileNotFoundException
34	SocketException
35	InterruptedIOException
36	ErrNotPoolingException
45	ErrTrnchkException
76	ErrServerTimedOutException
77	ErrClientTimedOutException
78	ErrNotTrnExtendException
79	ErrTrnchkExtendException
80	ErrNetDownAtServerException
81	ErrNetDownAtClientException
84	ErrInvalidMessageException
85	ErrBufferOverflowException
86	ErrCollisionMessageException
1001	IOException

Code	Exception name
`1002`	`UnknownHostException`

2.11.5 Method trace

TP1/Client/J outputs method trace information to the directory specified in the `TrcPath` argument of the `setMethodTraceMode` method or the directory specified in the `dcmethodtracepath` operand of the TP1/Client/J environment definition; the file names are `DCMTD1.TRC` and `DCMTD2.TRC`. You specify the file size with the `size` argument of the `setMethodTraceMode` method or the `dcmethodtracesize` operand of the TP1/Client/J environment definition.

Many methods are executed within a TP1Client class. A method trace outputs to the file the execution order and times of these internal methods. In a Java environment, only a limited amount of information can be obtained in the event of an error. You can use the method trace information about the internal processing resulting in an error to determine the cause of an error.

The output format of a method trace file is as follows:

```
yyyy/mm/dd hh:mm:ss.uuu Location = lll ThreadName = ttt
MethodName = nnnnnnnnnnnnnnnnnnnn
ADDRESS +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +a +b +c +d +e +f
0123456789abcdef
00000000 00 00 00 00 c7 0b 49 17 00 00 00 00 00 06 00 05
......|.........
```

- *yyyy*`/`*mm*`/`*dd* *hh*`:`*mm*`:`*ss*`.`*uuu*: Date and time the method trace was collected
- *lll*: Type of entry or exit point
- *ttt*: Name of the executing thread

Character string following `ADDRESS`: Name of the method that was called

2.11.6 Debug trace

TP1/Client/J normally collects debug trace information in its memory buffer and outputs the information to a debug trace file only when an exception occurs. TP1/Client/J outputs a debug trace to the `TP1ClientJ` directory under the home directory of the user executing the Java VM; the file name is `dcClt`*XXXXXXXXXXXX*`.dmp` (where *XXXXXXXXXXXX* is a time stamp).

When a debug trace file is to be output, if the total number of debug trace files exceeds the file count specified in the `dccltdbgtrcfilecount` operand of the TP1/Client/J environment definition, the file with the oldest update date is deleted. The existing debug trace files are deleted until the total number of debug trace files reaches (*file-count-specified-in-the dccltdbgtrcfilecount-operand* - 1).

If debug traces cannot be output to files, such as when a CUP is operated as a Java applet or when the deletion of old debug trace files fails, debug traces are output to the standard output. When debug trace files are output to the standard output, you can view the debug trace information by opening the Java console using a browser facility, for example.

The output format of a debug trace is as follows:

```
(ttt)yyyy/mm/dd hh:mm:ss.uuu MethodName = nnnnnnnnnnnnnnnnnnnn(ii)
Location = lll
ADDRESS +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +a +b +c +d +e +f
0123456789abcdef
00000000 00 00 00 00 c7 0b 49 17 00 00 00 00 00 06 00 05
......|.........
```

- *ttt*: Thread ID
- *yyyy/mm/dd hh:mm:ss.uuu*: Date and time the debug trace was collected
- *nnnnnnnnnnnnnnnnnnnn*: Name of the executing thread
- *ii*: Instance ID
- *lll*: Type of entry or exit point

Character string following `ADDRESS`: Debug trace information

2.11.7 Performance analysis trace

Performance analysis trace (*PRF trace*) information is the trace information of the Cosminexus Application Server. By collecting the performance analysis trace, you can determine the flow of the series of processes that occurred during UAP execution and the time that was taken, and collect the information necessary for analyzing performance. Furthermore, when an error occurs, you can determine how far the process had reached.

For an overview of performance analysis traces and details on how to use them, see the manuals *Cosminexus Function Description* and *Cosminexus System Operation Guide*. This section explains how to use performance analysis traces when you run TP1/Client/J on Cosminexus Application Server, the information collected in performance analysis traces, and trace collection points.

(1) Outputting the performance analysis trace on Cosminexus Application Server

When you run TP1/Client/J on Cosminexus Application Server, you can output the performance analysis trace on Cosminexus Application Server. The performance analysis trace is collected at the TP1/Client/J trace collection point. To collect the performance analysis trace, specify `dccltprftrace=Y` in the TP1/Client/J environment definition.

When outputting the performance analysis trace on Cosminexus Application Server, the information collected by TP1/Client/J increases the size of the performance analysis trace that is output. Therefore, you must take this increase into consideration when specifying a size for the performance analysis trace.

(2) Collating Cosminexus Application Server with OpenTP1 trace

When you issue to TP1/Server an RPC that uses the name service or an RPC that uses the scheduler direct facility, you can add to the RPC message to be sent to the scheduler identification information (an IP address, for example) that is essentially unique to each TP1/Client/J instance. The added information is output to the performance verification trace of OpenTP1. By collating this OpenTP1 performance verification trace with the performance analysis trace of Cosminexus Application Server, you can identify the series of operational flow between Cosminexus Application Server and OpenTP1. When you issue to a TP1/Server whose version is 07-02 or later an RPC that uses the remote API facility, you can add identification information (an IP address, for example) to the RPC message to be sent to the scheduler.

To add identification information to the OpenTP1 performance verification trace, specify `dccltprfinfosend=Y` in the TP1/Client/J environment definition. For details about the OpenTP1 performance verification trace, see the manual *OpenTP1 Description*.

(3) Information collected in the performance analysis trace

Information collected by TP1/Client/J is output as added information to the performance analysis trace. The information collected in the performance analysis trace and a trace output example are described below.

Information collected in the performance analysis trace

Node ID: `_J`*aa* (4-byte alphanumeric character)

aa: Randomly chosen two alphanumeric characters (numbers (0-9) or letters (A-Z or a-z))

Root communication serial number: *bbbbbbbb* (4-byte hexadecimal number)

bbbbbbbb: IP address

RPC communication serial number: *ccccdddd* (4-byte hexadecimal number)

cccc: Randomly chosen 2-byte hexadecimal number

dddd: Communication serial number

Example of outputting performance analysis trace on Cosminexus Application Server

```
...    OPT                    ASCII
...    Hexadecimal display  _Jaa/0xbbbbbbbb/0xccccdddd...
```

Output example in which identification information is added to the performance verification trace of OpenTP1

```
PRF: Rec Node: smpl Run-ID: 0x416f809d Process: 2612 Trace: 11
Event: 0x2002 Time: 2004/10/15 17:17:07 195.000.000 Server-name: _scd
Rc: *********** Client:  - 0xccccdddd     Server: **** Root: _Jaa - 0xbbbbbbbb
Svc-Grp: tst_svg                      Svc: echo
Trn: *
```

(4) Performance analysis trace collection point

This subsection explains the details of the trace collection point for the performance analysis trace of TP1/Client/J for each type of processing.

(a) Establishing and terminating connection with the RAP-processing listener

The following table shows the event IDs, trace collection points, additional information, and PRF trace collection levels used when establishing or terminating connection with the RAP-processing listener.

Table 2-45: Details of the trace collection points when establishing or terminating connection with the RAP-processing listener

Event ID	Number in the figure#1	Trace collection point	Additional information#2	Level
`0x9180`	1	Before establishing connection to the RAP-processing listener	Information on the request destination (host name, port number)	A
`0x9181`	2	After establishing connection to the RAP-processing listener (collected only when an error occurs)	Internal return code	A
`0x9182`#3	3	After sending or receiving RAP-processing listener connection request data	Internal return code	A
`0x9183`	4	Before sending or receiving RAP-processing listener disconnection request data	N/A	A

Event ID	Number in the figure#1	Trace collection point	Additional information#2	Level
`0x9184`	5	After sending or receiving RAP-processing listener disconnection request data	Internal return code	A

Legend:

A: Standard

N/A: Not applicable

#1

Corresponds to the number inside Figure 2-28.

#2

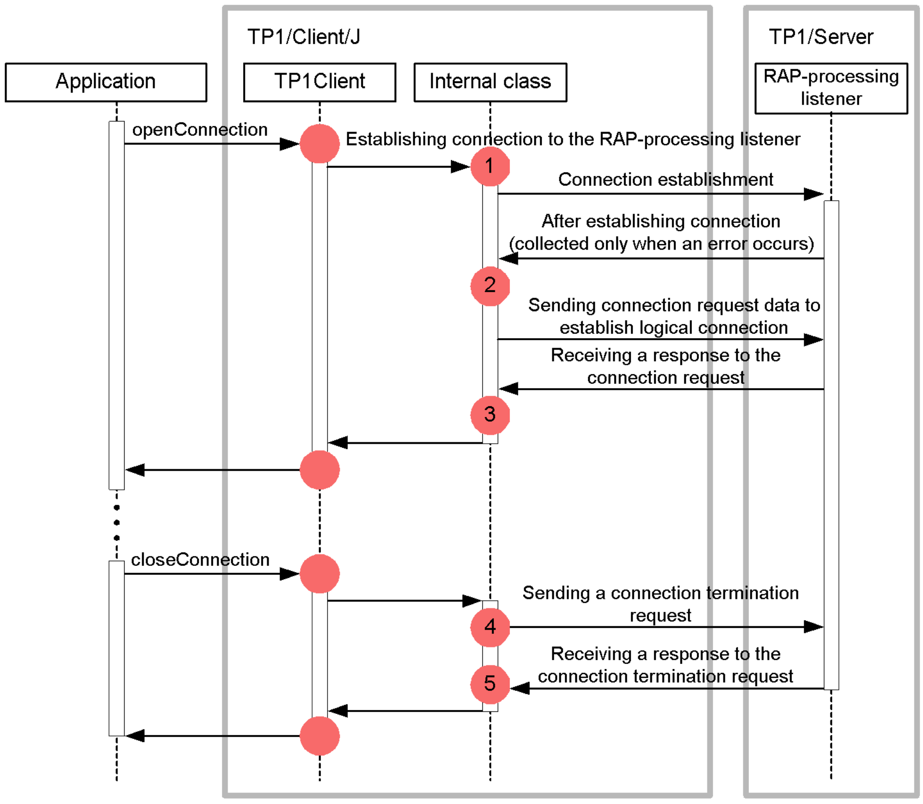
Information to be added to the performance analysis trace of Cosminexus Application Server

#3

A trace with this event ID is not collected if an error occurs in the connection establishment process and the event ID `0x9181` is collected.

The following figure shows the trace collection points in establishing or terminating connection with the RAP-processing listener:

Figure 2-33: Trace collection points in establishing or terminating connection with the RAP-processing listener


TP1/Client/J
TP1/Server
Application
TP1Client
Internal class
RAP-processing listener
openConnection
Establishing connection to the RAP-processing listener
1
Connection establishment
After establishing connection (collected only when an error occurs)
2
Sending connection request data to establish logical connection
Receiving a response to the connection request
3
closeConnection
Sending a connection termination request
4
Receiving a response to the connection termination request
5

Legend: ● : Indicates a trace collection point. The PRF trace collection level is *Standard*.

(b) API surrogate execution request by the remote API facility

The following table shows the event IDs, trace collection points, additional information, and PRF trace collection levels when an API surrogate execution request is issued by the remote API facility.

Table 2-46: Details of the trace collection points when an API surrogate execution request is issued by the remote API facility

Event ID	Number in the figure#1	Trace collection point	Additional information#2	Level
`0x9190`	1	Before processing a request for surrogate execution of `dc_rpc_call`	N/A	A

Event ID	Number in the figure[#1]	Trace collection point	Additional information[#2]	Level
`0x9191`	2	After processing a request for surrogate execution of `dc_rpc_call`	Internal return code	A
`0x9192`	1	Before processing a request for surrogate execution of `dc_trn_begin`	N/A	A
`0x9193`	2	After processing a request for surrogate execution of `dc_trn_begin`	Internal return code	A
`0x9194`	1	Before processing a request for surrogate execution of `dc_trn_chained_commit`	N/A	A
`0x9195`	2	After processing a request for surrogate execution of `dc_trn_chained_commit`	Internal return code	A
`0x9196`	1	Before processing a request for surrogate execution of `dc_trn_chained_rollback`	N/A	A
`0x9197`	2	After processing a request for surrogate execution of `dc_trn_chained_rollback`	Internal return code	A
`0x9198`	1	Before processing a request for surrogate execution of `dc_trn_unchained_commit`	N/A	A
`0x9199`	2	After processing a request for surrogate execution of `dc_trn_unchained_commit`	Internal return code	A
`0x919A`	1	Before processing a request for surrogate execution of `dc_trn_unchained_rollback`	N/A	A
`0x919B`	2	After processing a request for surrogate execution of `dc_trn_unchained_rollback`	Internal return code	A

Legend:

A: Standard

N/A: Not applicable

#1

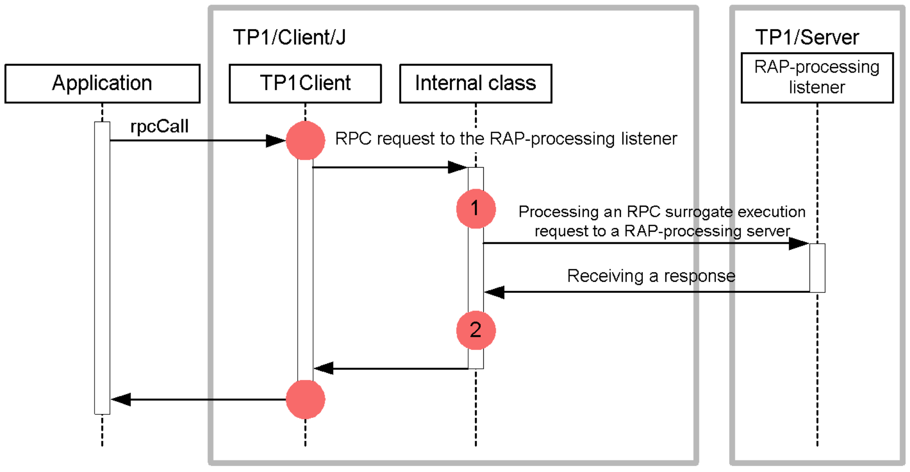
Corresponds to the number inside Figure 2-29.

#2

Information to be added to the performance analysis trace of Cosminexus Application Server

The following figure shows the trace collection points that are used when an API surrogate execution request is issued by the remote API facility:

Figure 2-34: Trace collection points that are used when an API surrogate execution request is issued by the remote API facility


TP1/Client/J
TP1/Server
Application
TP1Client
Internal class
RAP-processing listener
rpcCall
RPC request to the RAP-processing listener
1
Processing an RPC surrogate execution request to a RAP-processing server
Receiving a response
2

Legend: ● : Indicates a trace collection point. The PRF trace collection level is *Standard*.

(c) Issuing an RPC to the schedule server

The following table shows the event IDs, trace collection points, additional information, and PRF trace collection levels used when issuing an RPC to the schedule server:

Table 2-47: Details of the trace collection points when issuing an RPC to the schedule server

Event ID	Number in the figure#1	Trace collection point	Additional information#2	Level
`0x91C0`	1	Before establishing connection to the schedule server	Information on the host at the service request destination (host name, port number)	A
`0x91C1`	2	Before processing a service request to issue an RPC to the schedule server	Internal return code	A
`0x91C2`#3	3	After accepting a connection request to receive response from the SPP or schedule server	Internal return code#4	A

Event ID	Number in the figure#1	Trace collection point	Additional information#2	Level
`0x91C3`	4	Before establishing connection to the name server	Information on the host at the service request destination (host name, port number)	B
`0x91C4`	5	After executing a service information inquiry to the name server	Internal return code	B
`0x91C5`#3	6	After accepting a connection request to receive response data from the name server	Internal return code#5	B

Legend:

A: Standard

B: Detail

#1

Corresponds to the number inside Figures 2-30 and 2-31.

#2

Information to be added to the performance analysis trace of Cosminexus Application Server

#3

A trace with this event ID is collected when the transmission of the requested data is normally terminated.

#4

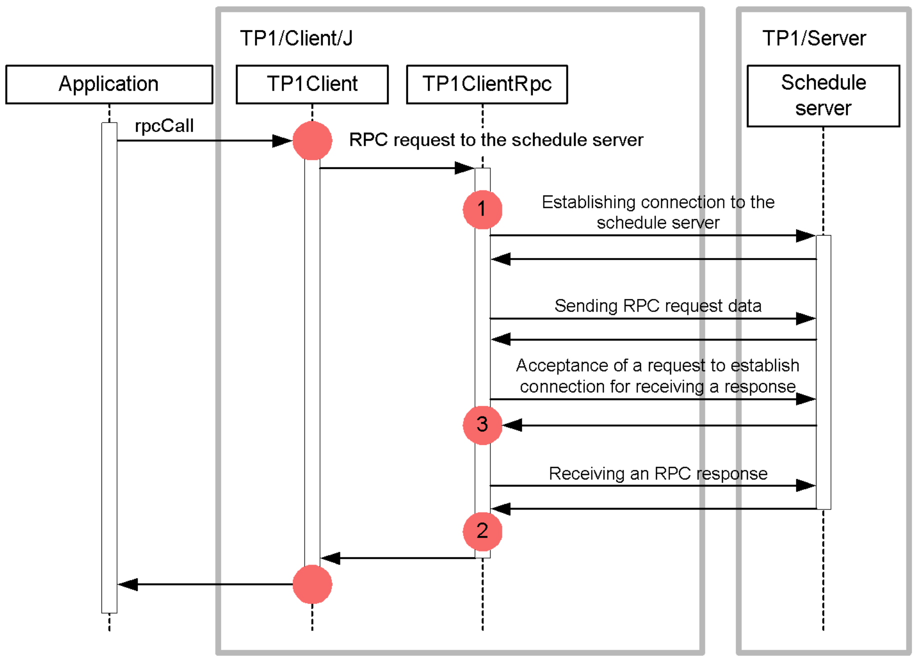
When accepting of the connection request to receive response data from the SPP or schedule server is terminated normally, the connection source information (IP address and port number) is also added.

#5

When accepting of the connection request to receive response data from the name server is terminated normally, the connection source information (IP address and port number) is also added.

The following figure shows the trace collection points that are used when issuing an RPC to the schedule server:

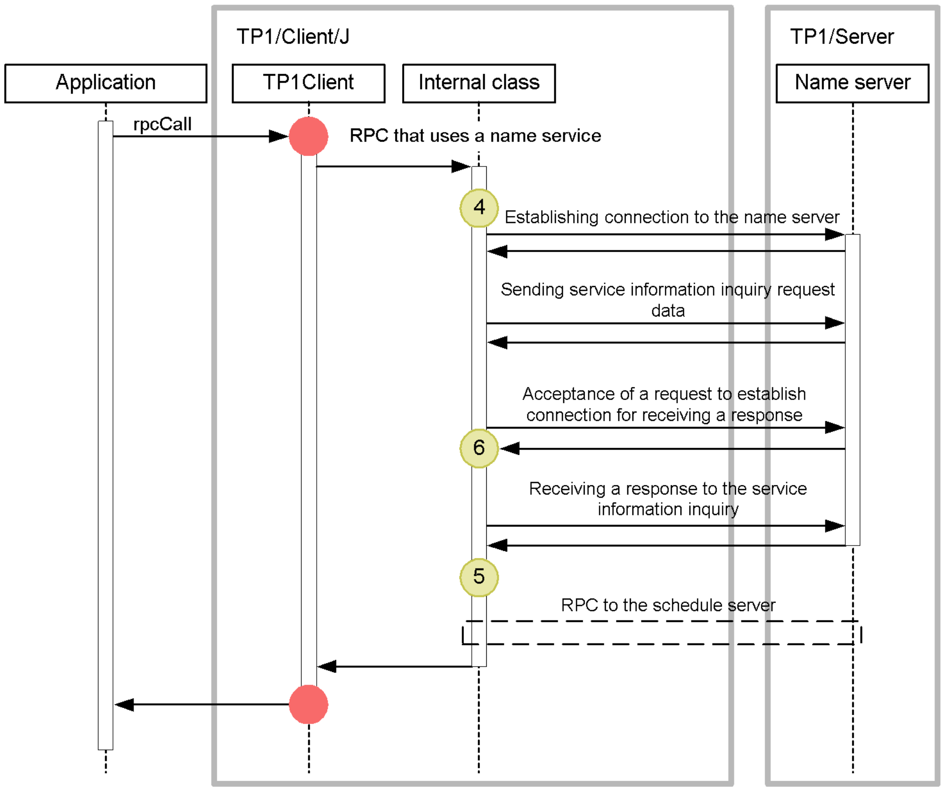
Figure 2-35: Trace collection points that are used when issuing an RPC to the schedule server


TP1/Client/J
TP1/Server
Application
TP1Client
TP1ClientRpc
Schedule server
rpcCall
RPC request to the schedule server
1
Establishing connection to the schedule server
Sending RPC request data
Acceptance of a request to establish connection for receiving a response
3
Receiving an RPC response
2

Legend: ● : Indicates a trace collection point. The PRF trace collection level is *Standard*.

The following figure shows the trace collection points that are used when issuing a service information inquiry to the name server:

Figure 2-36: Trace collection points that are used when issuing a service information inquiry to the name server


TP1/Client/J
TP1/Server
Application
TP1Client
Internal class
Name server
rpcCall
RPC that uses a name service
4
Establishing connection to the name server
Sending service information inquiry request data
Acceptance of a request to establish connection for receiving a response
6
Receiving a response to the service information inquiry
5
RPC to the schedule server

Legend: 🔴 : Indicates a trace collection point. The PRF trace collection level is *Standard*.

🟡 : Indicates a trace collection point. The PRF trace collection level is *Detail*.

(d) API execution

The following table shows the event IDs, trace collection points, additional information, and PRF trace collection levels used during API execution.

Table 2-48: Details of the trace collection points during API execution

Event ID	Trace collection point	Additional information#1	Level
`0x91D0`	Entry point to the `openConnection` method	N/A	A
`0x91D1`	Exit point from the `openConnection` method	Internal return code	A

Event ID	Trace collection point	Additional information[#1]	Level
`0x91D2`	Entry point to the `closeConnection` method	N/A	A
`0x91D3`	Exit point from the `closeConnection` method	Internal return code	A
`0x91D4`	Entry point to the `rpcCall` method	`inlen`	A
`0x91D5`	Exit point from the `rpcCall` method	`outlen`, Internal return code	A
`0x91D6`	Entry point to the `rpcCallTo` method	`inlen`	A
`0x91D7`	Exit point from the `rpcCallTo` method	`outlen`, Internal return code	A
`0x91D8`	Entry point to the `trnBegin` method	N/A	A
`0x91D9`	Exit point from the `trnBegin` method	Internal return code[#2]	A
`0x91DA`	Entry point to the `trnChainedCommit` method	N/A	A
`0x91DB`	Exit point from the `trnChainedCommit` method	Internal return code[#2]	A
`0x91DC`	Entry point to the `trnChainedRollback` method	N/A	A
`0x91DD`	Exit point from the `trnChainedRollback` method	Internal return code[#2]	A
`0x91DE`	Entry point to the `trnUnchainedCommit` method	N/A	A
`0x91DF`	Exit point from the `trnUnchainedCommit` method	Internal return code	A
`0x91E0`	Entry point to the `trnUnchainedRollback` method	N/A	A
`0x91E1`	Exit point from the `trnUnchainedRollback` method	Internal return code	A

Legend:

A: Standard

N/A: Not applicable

#1

Information to be added to the performance analysis trace of Cosminexus Application Server

#2

At the start of a transaction, the transaction global identifier and the transaction branch identifier are also added.

(5) Performance analysis trace identification information that is output to a UAP trace

TP1/Client/J outputs performance analysis trace identification information at the end of the UAP trace exit-point information in order to connect performance analysis trace information of TP1/Server and Cosminexus with the trace information of TP1/Client/J.

The following methods are provided for outputting performance analysis trace identification information to a UAP trace:

- `openConnection()`
- `openConnection(String host, int port)`
- `closeConnection()`
- `rpcCall(String group, String service, byte[] in_data, int[] in_len, byte[] out_data, int[] out_len, int flags)`
- `rpcCall(String group, String service, byte[] in_data, byte[] out_data, int flags)`
- `rpcCallTo(DCRpcBindTbl direction, String group, String service, byte[] in_data, int[] in_len, byte[] out_data, int[] out_len, int flags)`
- `trnBegin()`
- `trnChainedCommit()`
- `trnUnchainedCommit()`
- `trnChainedRollback()`
- `trnUnchainedRollback()`

Note that the performance analysis trace identification information is output regardless of the value of the `dccltprfinfosend` operand in the TP1/Client/J environment definition.

An example of a UAP trace to which performance analysis trace identification information is output is shown below. Bold type indicates the performance analysis trace identification information.

```
2008/09/01 16:37:24.698  Location = Out  ThreadName = main
MethodName = TP1Client.rpcCall
Exception  =
ADDRESS   +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +a +b +c +d +e +f
0123456789abcdef
00000000  00 00 00 1c 53 75 7a 75 6b 69 00 00 00 00 00 00
....Suzuki......
00000010  00 00 00 00 00 00 00 00 00 00 00 51 00 00 00
...........Q...
```

```
00000020  5f 4a 30 6d 2f 30 78 30 61 64 31 30 66 37 63 2f  _J0m/
0x0ad10f7c/
00000030  30 78 32 62 35 64 30 30 30 31                   0x2b5d0001
```

2.12 Data compression facility

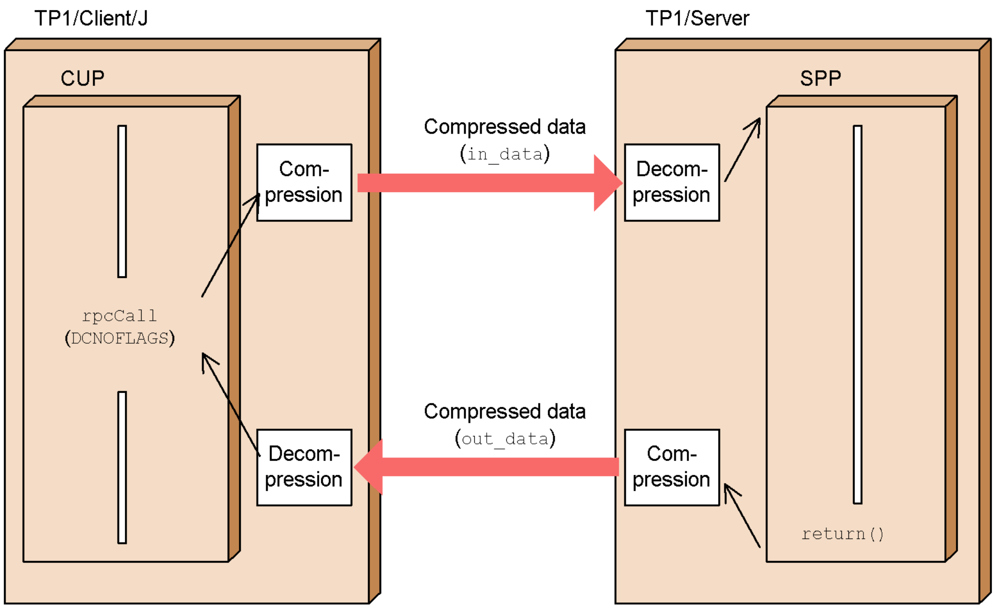
The *data compression facility* enables you to compress user data that is sent over the network by an RPC. You can use this facility to reduce the number of packets that are sent over the network, thereby reducing network congestion.

You use the `dccltdatacomp` operand in the TP1/Client/J environment definition to specify whether or not the data compression facility is used.

When this facility is used, TP1/Client/J compresses the value of the input parameter (`in_data`) set by the `rpcCall` method that is executed from a CUP. If compression is effective, the compressed data is then sent out over the network. The value of the response (`out_data`) to the query from SPP is also compressed by TP1/Server and then sent over the network. TP1/Client/J receives the response, restores the compressed data, and then passes it to the CUP.

The following figure provides an overview of the data compression facility.

Figure 2-37: Overview of database adapter



Note: In the case of TP1/Server version 03-06 or later, out_data is compressed by TP1/Server if compression would be effective, even when in_data was not compressed.

This facility is applicable when the version of TP1/Server at the service request destination is 03-03 or later.# It supports the RPC communication mode and all RPC interfaces. However, the following differences apply, depending on the TP1/Server version at the service request destination:

- For a TP1/Server version earlier than 03-06

  If the input parameter value is not compressed, the response value is not compressed by TP1/Server even if compression would be effective.

- For a TP1/Server version 03-06 or later

  The response value is compressed by TP1/Server if compression would be effective, even if the input parameter value was not compressed.

#

For an RPC using the remote API facility, the data compression facility is supported if the TP1/Server version is 03-05 or later.

The data compression facility cannot be used if the XA transaction linkage facility is used from uCosminexus TP1 Connector or Cosminexus TP1 Connector and if the TP1/Server version is 07-00 or earlier.

2.12.1 Effects of the data compression facility

The effects of the data compression facility depend on the contents of the user data. Use of this function is most effective when the user data contains many occurrences of the same characters in the same order. With some user data, use of the function has almost no effect.

Because there is some overhead associated with data compression and decompression, you should evaluate the advantages of data compression in light of its associated costs before you decide whether or not to use the data compression facility.

2.12.2 Notes about using the data compression facility

This subsection provides notes about using the data compression facility.

- If an RPC using the scheduler direct facility is issued and the TP1/Server version at the service request destination is earlier than 03-03, invalid data may be passed to SPP's service function. In such a case, do not issue to TP1/Server a service request that uses the data compression facility (specify `N` in the `dccltdatacomp` operand).

  If an RPC using the name service facility is issued and the TP1/Server version at the service request destination is earlier than 03-03, the data compression facility cannot be used even when it is set to be used, because this setting is ignored.

- If the size of the data obtained after compression would be greater than the size of the data before compression, or if the TP1/Server at the service request destination does not support the data compression facility, a message indicating that the data was not compressed is output to the error trace file of TP1/Client/J, and processing continues. In such a case, the data has not been compressed.

- The data compression facility cannot be used when you establish a permanent connection with a DCCM3 logical terminal and then issue a service request.
- If the `rpcCall` method returns an error, the service's response (`out_data`) is not guaranteed.
- The maximum length of an RPC message (maximum length of a user message that can be sent or received by the `rpcCall` method) is always the value set in the `dccltrpcmaxmsgsize` operand or the default value (1 megabyte), regardless of whether or not the data compression facility is used.
- If the XA transaction linkage facility is used from uCosminexus TP1 Connector or Cosminexus TP1 Connector, and if the version of the target TP1/Server is 07-00 or earlier, TP1/Client/J cannot use the data compression facility. If an attempt is made to use the data compression facility in such a case, the `KFCA32042-W` message may be output.

2.13 Source host specification facility

When you issue a connection establishment request using an RPC or the TCP/IP communication facility, TP1/Client/J enables you to specify the transmission source host for the CUP. This is called the *source host specification facility*.

If multiple network adapters are connected on the host where the CUP is running, the transmission source host to be used by the CUP when connection establishment is requested is determined by TCP/IP control. However, if you use the source host specification facility, you can fix the transmission source host to be used when the following connection establishment requests are issued:

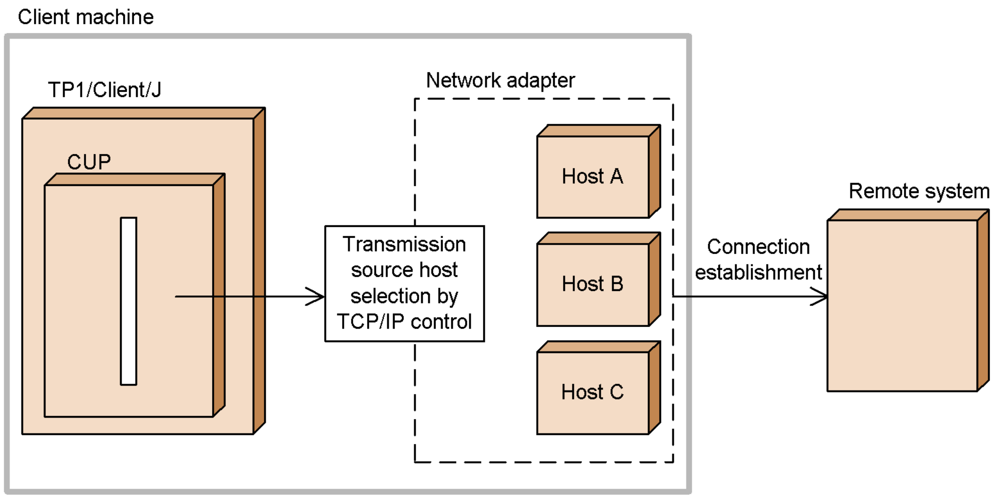
- RPC using the scheduler direct facility
- RPC using the name service
- RPC using the remote API facility
- RPC specifying the communication destination
- TCP/IP communication facility

To specify the transmission source host for a CUP, use the `dccltcupsndhost` operand in the TP1/Client/J environment definition.

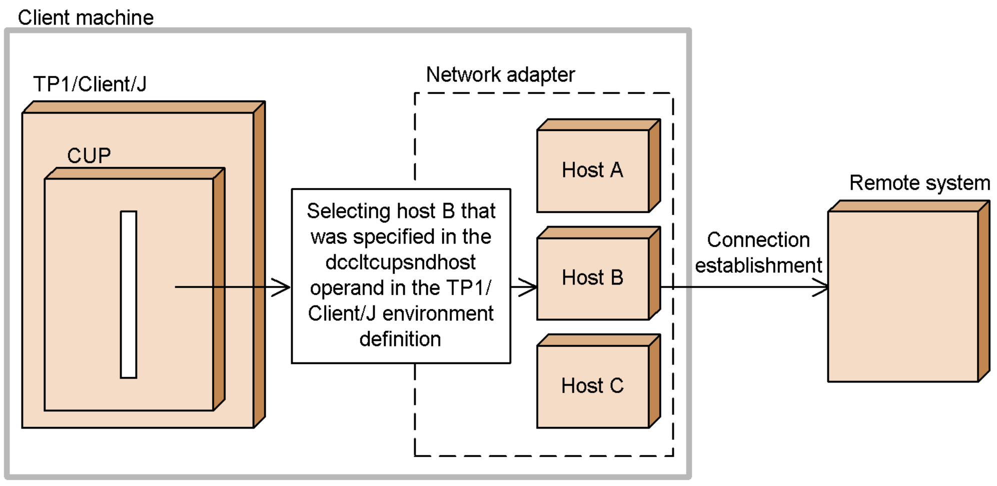
The following figure shows connection establishment when the source host specification facility is not used and when it is used.

Figure 2-38: When the source host specification facility is not used and when the facility is used

- When the source host specification facility is not used



- When the source host specification facility is used


Client machine
TP1/Client/J
CUP
Network adapter
Selecting host B that was specified in the dccltcupsndhost operand in the TP1/Client/J environment definition
Host A
Host B
Host C
Connection establishment
Remote system

2.14 Receive port fixing facility

TP1/Client/J allows you to fix the receive port for an RPC that uses the scheduler direct facility and an RPC that uses the name service. This is called the *receive port fixing facility*.

When you send an RPC's response from TP1/Server to TP1/Client/J, you use this facility to filter out notifications other than those approved for the receive port of TP1/Client/J by means of a firewall positioned between TP1/Server and TP1/Client/J.

To use this facility, specify the `dccltcuprcvport` operand in the TP1/Client/J environment definition.

The following describes when the receive port fixing facility is not used and when this facility is used.

(1) When the receive port fixing facility is not used

There is no filtering for the receive port in the RPC's response communication. The OS automatically allocates an undefined port as the receive port for the RPC in TP1/Client/J.

The following figures show when the receive port fixing facility is not used.

Figure 2-39: When the receive port fixing facility is not used (RPC using the scheduler direct facility)

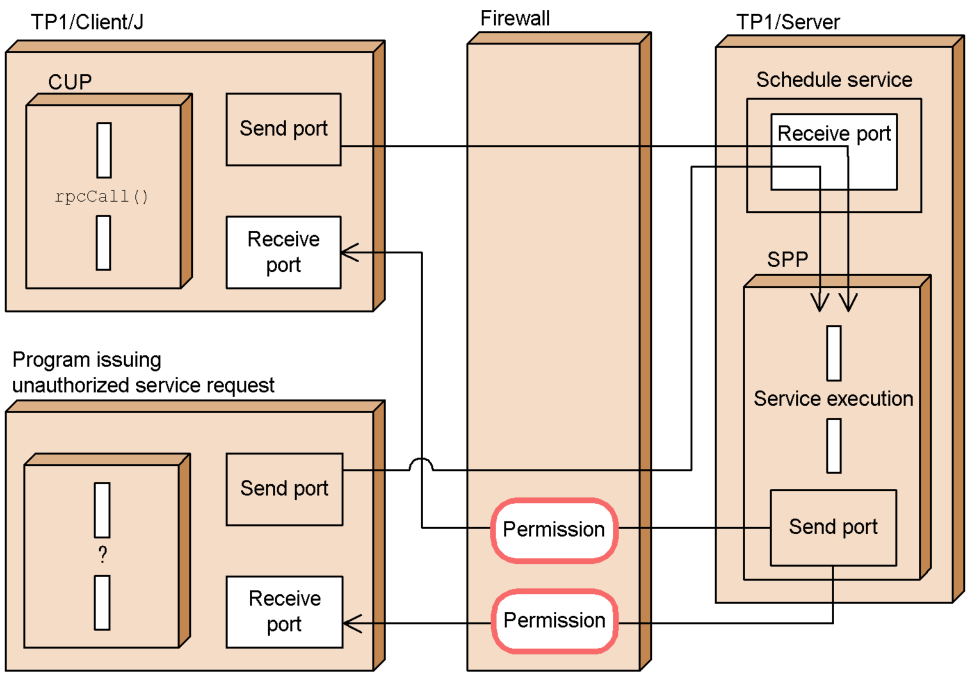
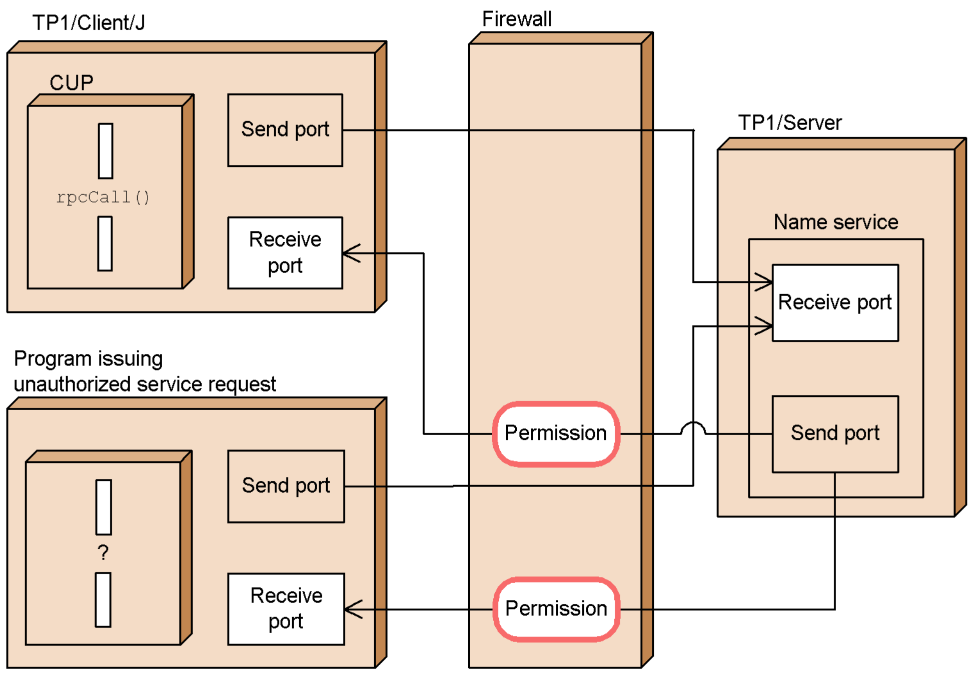


Figure 2-40: When the receive port fixing facility is not used (RPC using the name service)



(2) When the receive port fixing facility is used

The port specified in the `dccltcuprcvport` operand in the TP1/Client/J environment definition is used as the receive port for response communication to the RPC, and other communication is subject to filtering. This method enables you to use the firewall to filter response communications from TP1/Server for unauthorized service requests.

The following figures show when the receive port fixing facility is used.

Figure 2-41: When the receive port fixing facility is used (RPC using the scheduler direct facility)

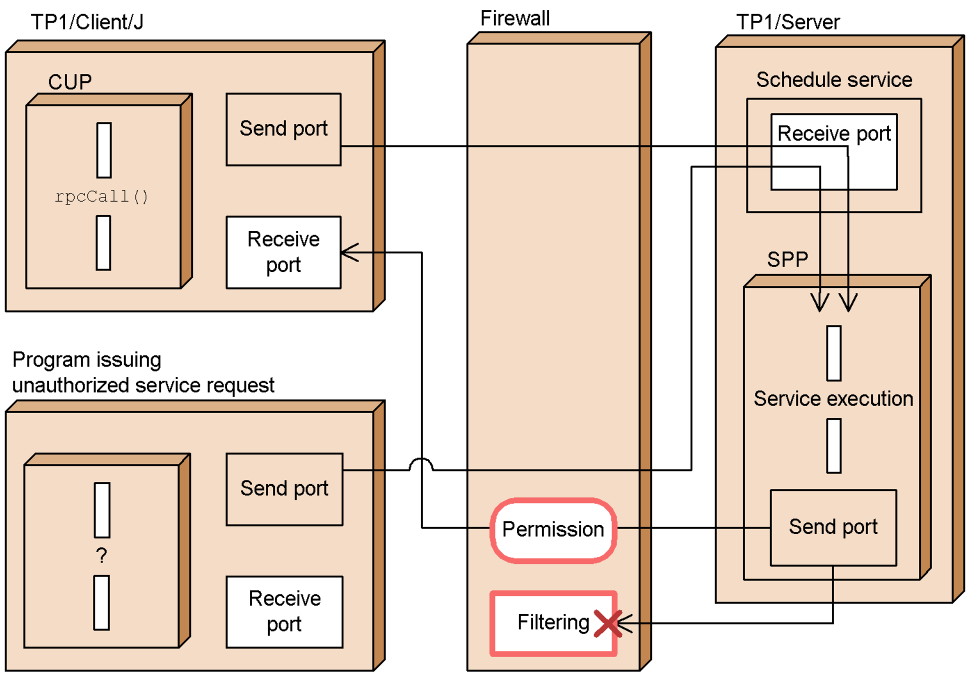
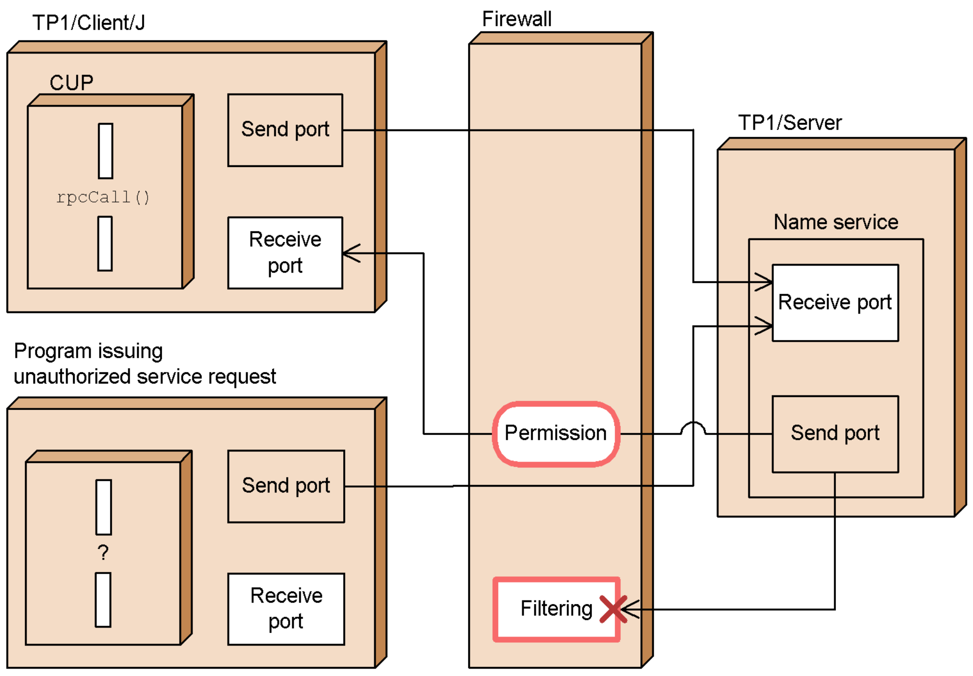


Figure 2-42: When the receive port fixing facility is used (RPC using the name service)



Chapter

3. Program Interface

This chapter describes the program interface for using TP1/Client/J.

3.1 List of APIs
3.2 How to use APIs

3.1 List of APIs

The following tables list the APIs that are used by TP1/Client/J.

Table 3-1: List of APIs (package and classes)

Classification	Name	Description or function
Package	`JP.co.Hitachi.soft.OpenTP1`	Package name of the TP1/Client/J classes
Class	`TP1Client`	Class name of TP1/Client/J
	`DCRpcBindTbl`	`DCRpcBindTbl` class

Table 3-2: List of APIs (methods)

Classification	Method	Description or function
Execution	`acceptNotification(byte[] inf,int[] inf_len,int port,int timeout,byte[] hostname,byte[] nodeid)`	Receives messages that are sent by server's function (`dc_rpc_cltsend` function).
	`acceptNotificationChained(byte[] inf,int[] inf_len,int timeout,byte[] hostname,byte[] nodeid)`	
	`cancelNotification(byte[] inf,int inf_len,String hostname,int port)`	Releases the unidirectional server message reception wait status (issuance of the `acceptNotification` or `acceptNotificationChained` method).
	`closeConnection()`	Releases a permanent connection between a CUP and a RAP-processing listener and server.
	`closeNotification()`	Deletes an environment for using the unidirectional message consecutive reception facility.
	`cltAssemSend(byte[] buff, int sendleng, String hostname, int portnum, int timeout, int flags)`	Uses the receive message assembly facility to send a message.

Classification	Method	Description or function
	`cltAssemReceive(byte[] buff, int[] recvleng, int timeout, int flags)`	Uses the receive message assembly facility to receive a message.
	`closeConnection()`	Releases a permanent connection between a CUP and a RAP-processing listener and server.
	`cltReceive(byte[] buff, int[] recvleng, int timeout, int flags)`	Receives a message sent by an MHP.
	`cltSend(byte[] buff, int sendleng, String hostname, int portnum, int flags)`	Sends a message to an MHP.
	`getTrnID(byte[] gid, byte[] bid)`	Obtains the transaction global identifier and transaction branch identifier.
	`openConnection()`	Establishes a permanent connection with a RAP-processing server or connects to TP1/Web and starts a virtual session.
	`openConnection(String host, int port)`	
	`openConnection(String url, short flags)`	
	`openNotification(int port)`	Creates an environment for using the unidirectional message consecutive reception facility.
	`rpcCall(String group, String service, byte[] in_data, int[] in_len, byte[] out_data, int[] out_len, int flags)`	Issues an RPC to the specified *service-group-name* + *service-name*.
	`rpcCall(String group, String service, byte[] in_data, byte[] out_data, int flags)`	
	`rpcCallTo(DCRpcBindTbl direction, String group, String service, byte[] in_data, int[] in_len, byte[] out_data, int[] out_len, int flags)`	Issues an RPC to the *service-group-name* + *service-name* of the specified communication-target node.
	`rpcClose()`	Releases the environment for calling a TP1/Server SPP.

Classification	Method	Description or function
	`rpcOpen()`	Initializes the environment for calling a TP1/Server SPP.
	`rpcOpen(String deffilename)`	
	`trnBegin()`	Starts a global transaction.
	`trnChainedCommit()`	Acquires the synchronization point of a transaction.
	`trnChainedRollback()`	Rolls back a transaction.
	`trnInfo()`	Determines whether or not the TP1Client object is running as a transaction.
	`trnUnchainedCommit()`	Acquires the synchronization point of a transaction.
	`trnUnchainedRollback()`	Rolls back a transaction.
Definition	`setConnectInformation(byte[] inf, short inf_len)`	Sets terminal identification information.
	`setDataTraceMode(String TrcPath, int size, int Datasize, boolean flag)`#	Specifies whether or not to output to a file a data trace when an RPC is issued from a Java application or servlet.
	`setDccltdelay(int sec)`	Sets the communication delay time between a CUP and a RAP-processing listener and server.
	`setDccltextend(int flags)`	Sets the extension level of the TP1/Client/J facility. The user can specify multiple extension levels by specifying the disjunction of each value.
	`setDccltinquiretime(int sec)`	Sets the maximum time for the inquiry interval that is monitored from the RAP-processing server (if no service request is received from the client within this period, the connection is released).
	`setDchost(String host, int port)`	Sets the host name and port number of the TP1/Server as a gateway.
	`setDcselint(int msec)`	This method serves no purpose. It is retained only to provide compatibility with the source of the previous version.
	`setDcwatchtim(int sec)`	For a synchronous-response RPC, sets the maximum time to wait for a response from the service since a service request was issued from the CUP to the SPP.

Classification	Method	Description or function
	`setErrorTraceMode(String TrcPath, int size, boolean flag)`#	Specifies whether or not to output to a file information about connection establishment with the server, disconnection from the server, and error trace when the TP1Client class is called from a Java application or servlet.
	`setMethodTraceMode(String TrcPath, int size, boolean flag)`#	Specifies whether or not to output to a file a method trace when the TP1Client class is called from a Java application or servlet.
	`setRpcextend(int extendoption)`	Changes RPC-related settings.
	`setScdDirectObject(String scdhost, int scdport, int flags)`	Sets the host name and port number of the communication-target scheduler.
	`setTraceArray(String[] array)`	Specifies the array (memory trace) for storing information about connection establishment with the server, disconnection from the server, and error trace when the TP1Client class is called from a Java application, servlet, or applet. To view the error information, you must use a Java application, servlet, or applet to reference the contents of this array.
	`setUapTraceMode(String TrcPath, int size, boolean flag)`#	Specifies whether or not to output to a file a UAP trace when the TP1Client class is called from a Java application or servlet.

#

Only Java applications and servlets can use this method. If an attempt is made to use this method with a Java applet, `SecurityException` occurs.

3.2 How to use APIs

After creating a Java applet, application, or servlet using the class library provided by TP1/Client/J, you can manipulate TP1/Client/J, mainly by using five APIs.

1. Preparing to use TP1/Client/J
2. Creating an instance of the TP1Client class
3. Initializing the RPC environment
4. Connecting to TP1/Server's RAP-processing server (applicable if RPCs are issued using the remote API facility)
5. Calling a remote service (RPC)
6. Releasing the permanent connection with the RAP-processing listener or server (applicable if RPCs are issued using the remote API)
7. Releasing the RPC environment

This section provides an overview and examples of the APIs in the order they are executed. For details about the classes, see *4. Classes Used with TP1/Client/J*.

Note:

Methods of the `TP1Client` class are not thread-safe. If you execute a method of an instance of the same `TP1Client` class from multiple threads, make sure that you apply a lock so that they will not be executed concurrently. If an attempt is made to execute a method of an instance of the same `TP1Client` class from multiple threads, the corresponding method may return an unexpected error.

3.2.1 Execution order of APIs

(1) Preparing to use TP1/Client/J

Declare import of the TP1/Client/J package in the program.

Import the TP1/Client/J package in the same manner as when the package of each Java facility is imported.

```
import java.applet.*;                  // Import java.applet package
import java.awt.*;                     // Import java.awt package
import JP.co.Hitachi.soft.OpenTP1.*;   // Import TP1Client package
```

(2) Creating an instance of the TP1Client class>

The class name of TP1/Client/J is `TP1Client`. Create an instance of the `TP1Client` class when you need to access TP1/Server.

```
TP1Client stock;           // Declare the instance variable named stock
stock = new TP1Client();   // Create an instance of the TP1Client class
                           // (constructor call)
```

(3) Initializing the RPC environment

Initialize the RPC environment to call TP1/Server's SPP. To initialize the RPC environment, call the `rpcOpen` method.

If an error occurs during the call, the `rpcOpen` method returns an exception. Execute exception processing, if necessary.

If you are issuing an RPC that uses the scheduler direct facility, an RPC that uses the name service, or an RPC specifying the communication target, be sure to call the `rpcOpen` method. If you are issuing an RPC that uses the remote API facility, you need not call the `rpcOpen` method.

When acquiring the TP1/Client/J environment definition from the system properties

```
try {
  stock.rpcOpen();                // Initialize the RPC environment
} catch(ErrFatalException e) {    // Occurrence of RPC environment
                                  // initialization error
  System.out.println("CUP Initialize error");
} catch (TP1ClientException e) {  // Occurrence of other error
  System.out.println("error occured");
}
```

When acquiring the TP1/Client/J environment definition from a file

```
try {
  stock.rpcOpen("C:\\Clt4J\\conf\\clt4j.ini");
                                  // Initialize the RPC environment
} catch(ErrFatalException e) {    // Occurrence of RPC environment
                                  // initialization error
  System.out.println("CUP Initialize error");
} catch (TP1ClientException e) {  // Occurrence of other error
  System.out.println("error occured");
}
```

(4) Connecting to TP1/Server's RAP-processing server (applicable if RPCs are issued using the remote API facility)

To issue an RPC that uses the remote API facility, connection is established with TP1/Server's RAP-processing listener and server. A dedicated TCP/IP connection is established between the Java applet, application, or servlet and the RAP-processing server until the connection is released.

To establish a connection, call the `openConnection` method of the `TP1Client` class. If an error occurs during the call, the `openConnection` method returns an exception. Execute exception processing, if necessary.

If you are issuing an RPC that uses the scheduler direct facility, an RPC that uses the name service, an RPC specifying the communication target, or an RPC that uses the auto connect mode, do not call the `openConnection` method.

```
try {
  stock.openConnection("stocksv",12000);
                                    // Establish connection with RAP-processing
                                    // listener and server
} catch(ErrTimedOutException e) {  // Occurrence of connection timeout
  System.out.println("server inactive");
} catch (TP1ClientException e) {  // Occurrence of other error
  System.out.println("error occured");
}
```

The CUP may not be able to detect release of the permanent connection if the RAP-processing listener, RAP-processing server, or OpenTP1 system terminates abnormally while the permanent connection is established. In such a case, the next method issued (`rpcCall` or `closeConnection`) may return `ErrTimedOutException`.

(5) Calling a remote service (RPC)

If connection is established successfully with the RAP-processing server, you can call (RPC) a remote service (SPP). To issue an RPC, call the `rpcCall` method. In the arguments of the `rpcCall` method, specify the service group name for the service, the service name, inquiry data, inquiry data length, response data reception area, response data reception area size, and RPC execution mode.

If an error occurs during the call, the `rpcCall` method returns an exception. Execute exception processing, if necessary.

```
int in_len[] = new int[1];
int out_len[] = new int[1];
byte in_data[];
byte out_data[];
String in = "PN=0012-2456-12";   // Inquiry data (fixed character string
                                             //   for convenience)
in_len[0] = 16;          // Inquiry data length (fixed string for convenience)
out_len[0] = 16;         // Response data area size (fixed for convenience)
in_data = new byte[in_len[0]];     // Create inquiry data area
in.getBytes(0, in.length(), in_data, 0);  // Set inquiry data
out_data = new byte[out_len[0]];  // Create response data area
try {
  stock.rpcCall("group", "service",   // Execute RPC (inquiry
                                      //   response type)
      in_data, in_len, out_data, out_len, stock.DCNOFLAGS);
} catch (ErrTimedOutException e) {    // RPC request timeout
  System.out.println("RPC timedout");
} catch (TP1ClientException e) {      // Occurrence of other error
  System.out.println("error occured");
}
```

There are three types of RPCs, as listed in the table below; specify the desired type in the `flags` argument:

RPC type		Function	Specification method
Inquiry response	Without chain	A response is returned after a service request is issued to the server. The next time an inquiry is sent to the same server, the same server process as the previous one may not be used.	`DCNOFLAGS`
	With chain	A response is returned after a service request is issued to the server. The next time an inquiry is sent to the same server, the same server process as the previous one is used (the server process is locked). This is not applicable to RPCs that use the scheduler direct facility or name service.	`DCRPC_CHAINED`
No response		No response can be received after a service request is issued to the server. The client does not know whether the service executed successfully. Because the RPC does not wait for a response, the client's execution performance is high.	`DCRPC_NOREPLY`

Note

When you have used `DCRPC_CHAINED` to call services, you must execute an RPC specifying `DCNOFLAGS` when calling the last service to indicate the end of the chained RPCs.

(6) Releasing permanent connection with the RAP-processing listener or server (applicable if RPCs are issued using the remote API)

Once an online application with TP1/Server has been completed, call the `closeConnection` method to release the permanent connection from the

RAP-processing listener and server. This method has no arguments.

If an error occurs during the call, the `closeConnection` method returns an exception. Execute exception processing, if necessary.

```
try {
  stock.closeConnection();
           // Release permanent connection from RAP-processing listener and server
} catch (TP1ClientException e) {  // Occurrence of error
  System.out.println("error occured");
}
```

(7) Releasing the RPC environment

At the end of the CUP, call the `rpcClose` method to release the RPC environment of the CUP. This method has no arguments.

If an error occurs during the call, the `rpcClose` method returns an exception. Execute exception processing, if necessary.

```
try {
  stock.rpcClose();               // Release RPC environment
} catch (TP1ClientException e) {  // Occurrence of error
  System.out.println("error occured");
}
```

3.2.2 Adjustment during TP1/Client/J execution

Normally, method monitoring intervals are set in the TP1/Client/J environment definition. These specifications in the TP1/Client/J environment definition apply to all methods. If a particular method requires a considerable amount of processing time on the TP1/Server system, you will have to specify large values, making it impossible to adjust the execution time according to the processing.

To solve this problem, TP1/Client/J provides methods that enable you to adjust the method monitoring intervals dynamically. When one of these methods is executed, the value specified in its argument is set as the applicable value during execution and remains in effect as long as the instance of the `TP1Client` class exists (until the `rpcClose` method is called) or until the same method is called with a different argument value specified. For details about these methods, see *4. Classes Used with TP1/Client/J*.

(1) Adjusting the time monitoring

Three methods are provided for adjusting time monitoring: `setDcwatchtim`, `setDccltinquiretime`, and `setDccltdelay`. The time monitoring that can be specified with each method is described below.

`setDcwatchtim` method

Sets the response monitoring interval. When the `openConnection`, `rpcCall`, and `closeConnection` methods are called, inquiry response takes place between the CUP and the RAP-processing listener and server. A timeout occurs if no response is returned within the specified response monitoring interval since the inquiry began. Specify an appropriate value taking into account the duration of service execution and the communication speed.

`setDccltinquiretime` method

The permanent connection between a CUP and a RAP-processing listener and server that is established by calling the `openConnection` method remains established until the `closeConnection` method is called. If TP1/Client/J cannot call the `closeConnection` method due to an error, the RAP-processing listener and server will not be released. To avoid this, you use this method to set the length of time allowed for a response from TP1/Client/J before connection with the RAP-processing listener and server is released forcibly.

Note that if the connection-target RAP-processing server is DCCM3, the value set in the `setDccltinquiretime` method is ignored.

`setDccltdelay` method

When the `rpcCall` method is called, the TP1/Client/J system monitors the response time based on the value specified in the `setDcwatchtim` method. The RAP-processing server that relays RPC requests from TP1/Client/J monitors for a response for the specified amount of time (i.e., the value specified in the `setDcwatchtim` method minus the value specified in the `setDccltdelay` method). The value specified in the `setDccltdelay` method is a delay time for communication between TP1/Client/J and TP1/Server. If the monitoring interval is the same for both TP1/Client/J and the RAP-processing server, a timeout occurs at TP1/Client/J before a timeout occurs at the RAP-processing server because TP1/Client/J starts response monitoring before the RAP-processing server starts response monitoring. If the RAP-processing server sends a response just in time, a timeout may have already occurred at TP1/Client/J and the response may not be received. To prevent such an event, this method is used to force a timeout at the RAP-processing server. To put the value specified in this method into effect, either specify `dcwatchtiminherit=Y` in the TP1/Client/J environment definition or specify the `DRPC_WATCHTIMINHERIT` argument in the `setRpcextend` method.

3.2.3 Collecting error information and adjusting facilities

Three methods are provided for collecting error information and adjusting facilities: `setTraceArray`, `setDccltextend`, and `setRpcextend`. The value specified in each method takes effect when the method is called. When these methods are called, the values specified in their arguments take effect during execution and remain in effect as long as the instance of the `TP1Client` class exists (until the `rpcClose` method is called) or until the same method is called with a different argument value specified.

`setTraceArray` method

For Java applets, you cannot create a file in its execution environment due to Java security limitations. This makes it impossible to collect error information as trace information in files. In such a case, use the memory trace facility to collect trace information. If you specify a `String` type array with the `setTraceArray` method, the `TP1Client` class stores error information in the specified array. In the event of a Java applet or application error, you can view the contents of the `String` array with a browser in order to determine the cause of the error and take appropriate action. For details about memory traces, see *2.11.4 Error trace and memory trace*.

`setDccltextend` method

Specifies whether or not you wish to use a facility that depends on the execution environment. You specify whether or not the `dc_rpc_get_callers_address` function of the SPP in the TP1/Server system is to be used to view the IP address of TP1/Client/J. With Microsoft Internet Explorer 3.0J, you cannot check the local IP in the TP1/Client/J system. In a Windows 95 multi-homed host environment, this operation results in an internal OS error. Therefore, the default is that the local IP is not checked.

`setRpcextend` method

Specifies functional extension options of RPCs that are issued from TP1/Client/J. To specify multiple extension levels of the RPC facility, specify the disjunction of each value.

3.2.4 Instructing trace output

To obtain various traces, specify necessary information either in the TP1/Client/J environment definition or in the applicable methods during execution.

In the case of Java applets, you cannot collect traces due to security limitations. In the case of Java servlets, security limitations depend on the application server being used; for details, see the documentation for your application server. The four methods for changing the settings for whether or not traces are to be obtained during execution are `setUapTraceMode`, `setDataTraceMode`, `setErrorTraceMode`, and `setMethodTraceMode`.

For details about the contents of the trace files, see *2.11.1 Contents of trace files*. The specified traces are output when the following methods are executed.

`setUapTraceMode` method

Specifies that information about the call to each method of the `TP1Client` class is to be output to a file.

`setDataTraceMode` method

Specifies that data on communication with TP1/Server is to be output to a file.

`setErrorTraceMode` method

Specifies that information on errors is to be output to a file.

`setMethodTraceMode` method

Specifies that activities of internal methods of the `TP1Client` class are to be output to a file. Because there is little available error information in a Java environment, this information is used to determine the processing that resulted in an error.

Chapter

4. Classes Used with TP1/Client/J

This chapter describes the classes that are used with TP1/Client/J.

Class TP1Client

```
java.lang.Object
  └── JP.co.Hitachi.soft.OpenTP1.TP1Client
```

```
public final class TP1Client
extends java.lang.Object
```

This class issues a service request to the TP1/Server server.

Related item

`TP1ClientException`

Fields

- `DCNOFLAGS`
  `public static final int DCNOFLAGS = 0x00000000`

  Specifies synchronous-response RPC as the RPC mode.

- `DCRPC_NOREPLY`
  `public static final int DCRPC_NOREPLY = 0x00000001`

  Specifies non-response type RPC as the RPC mode.

- `DCRPC_CHAINED`
  `public static final int DCRPC_CHAINED = 0x00000004`

  Specifies chained RPC as the RPC mode.

- `DCRPC_SCD_LOAD_PRIORITY`
  `public static final int DCRPC_SCD_LOAD_PRIORITY = 0x00000008`

  Specifies whether or not the TP1/Server that is the gateway that accepts a service request has priority for load balancing.

- `DCRPC_WATCHTIMINHERIT`
  `public static final int DCRPC_WATCHTIMINHERIT = 0x00000010`

  Specifies whether or not the RAP-processing server is to inherit the CUP's maximum time to wait for a response when an RPC is issued using the remote API facility.

- `DCRPC_RAP_AUTOCONNECT`
  `public static final int DCRPC_RAP_AUTOCONNECT = 0x00000020`

Specifies whether or not TP1/Client/J is to implement automatic connection establishment when an RPC is issued using the remote API facility.

- `DCRPC_TPNOTRAN`
  ```
  public static final int DCRPC_TPNOTRAN = 0x00000020
  ```

  Specifies RPC that does not inherit transactions as the RPC mode.

- `DCRPC_WATCHTIMRPCINHERIT`
  ```
  public static final int DCRPC_WATCHTIMRPCINHERIT = 0x00000040
  ```

  Specifies whether or not the server is to inherit the CUP's maximum time to wait for a response.

- `DCRPC_MAX_MESSAGE_SIZE`
  ```
  public static final int DCRPC_MAX_MESSAGE_SIZE = 1048576
  ```

  Specifies the maximum RPC message length. The permitted maximum value is 1048576 bytes. However, when the `dccltrpcmaxmsgsize` operand is used, the maximum RPC message length is the value specified by the `dccltrpcmaxmsgsize` operand rather than the value specified in `DCRPC_MAX_MESSAGE_SIZE`.

Constructors

- `TP1Client`
  ```
  public TP1Client()
  ```

  Creates an instance of the `TP1Client` class that issues an RPC to the TP1/Server server. Use this constructor to create a CUP as a Java application or servlet.

- `TP1Client`
  ```
  public TP1Client(Applet app)
  ```

  Creates an instance of the `TP1Client` class that issues an RPC to the TP1/Server server. Use this constructor to create a CUP as a Java applet.

  Parameter

  `app`

  Specifies an instance object of a Java applet.

Methods

- `setDccltinquiretime`
  ```
  public void setDccltinquiretime(int sec)
                              throws ErrInvalidArgsException
  ```

  Sets the maximum time from one inquiry to another that is issued by a CUP to the server. This is the timer value that is monitored by the RAP-processing server. If there is no inquiry within the specified time, the RAP-processing server forcibly releases the permanent connection. You can call this method while an instance of the `TP1Client` class exists.

Parameter

`sec`

Specifies the maximum time of the inquiry interval that is to be monitored by the RAP-processing server, expressed in the range from 0 to 1048575 (seconds). If you specify 0, the RAP-processing server waits indefinitely for an inquiry from the CUP.

Return value

None.

Exception

`ErrInvalidArgsException`

The value specified in the `sec` argument is not within the range from 0 to 1048575.

■ `setDccltdelay`

```
public void setDccltdelay(int sec)
                   throws ErrInvalidArgsException
```

Sets the delay time for communication between the RAP-processing server and the CUP. By specifying this method, you can end monitoring of the maximum time to wait for a response in the RAP-processing server system early by the specified amount of time and avoid missing a message that may arrive after a timeout occurs in the CUP system.

If you specify 0 in the `setDcwatchtim` method as the maximum time to wait for a response, the communication delay time specified in this method is ignored. If the maximum time to wait for a response minus the RAP-processing server's communication time is 0 or a negative value, the RAP-processing server assumes 1 as the maximum time value for waiting for a response in the RAP-processing server system.

To use the value specified in this method, specify either `dcwatchtiminherit=Y` in the TP1/Client/J environment definition or the `DRPC_WATCHTIMINHERIT` argument in the `setRpcextend` method.

Parameter

`sec`

Specifies the delay time for communication between the RAP-processing server and the CUP, expressed in the range from 0 to 65535 (seconds).

Return value

None.

Exception

`ErrInvalidArgsException`

The value specified in the `sec` argument is not within the range from 0 to 65535.

■ `setDcwatchtim`
```
public void setDcwatchtim(int sec)
                   throws ErrInvalidArgsException
```

For synchronous-response RPCs, this method sets the maximum time to wait for a response from the service since a service request was sent from the CUP to the SPP. The value specified with this method is also used as the maximum time to wait for a response for internal communications of TP1/Client/J.

Parameter

`sec`

Specifies the maximum time to wait for a response, expressed in the range from 0 to 65535 (seconds). If you specify 0, the CUP waits for a response indefinitely. Depending on the Java environment in use, release of a TCP/IP connection may not be detected even if the server is shut down. If this value were set to 0 in such a case, the CUP would wait indefinitely for a response from the nonexistent server; for this reason, you should specify an appropriate wait time.

Return value

None.

Exception

`ErrInvalidArgsException`

The value specified in the `sec` argument is not within the range from 0 to 65535.

■ `setDcselint`
```
public void setDcselint(int msec)
                 throws ErrInvalidArgsException
```

Use of this method is not recommended. It is retained only to provide compatibility with the source of the previous version. This method serves no purpose.

■ `setDccltextend`
```
public void setDccltextend(int flags)
```

Specifies the extension level of TP1/Client/J facilities. To specify multiple extension levels, specify the disjunction of each value.

Parameter

`flags`

Specifies the extension level of the following facility:

`0x00000000`: Does not extend the TP1/Client/J facility.

`0x00000001`: Extends the TP1/Client/J facility. When the `rpcCall` method is called, the IP address of the local CUP is reported to the service. Specify this value if you need to obtain the address of the CUP by using the called service to execute the `dc_rpc_get_callers_address()` function.

Return value

None.

■ `setRpcextend`
```
public void setRpcextend(int extendoption)
                  throws ErrInvalidArgsException
```

Specifies functional extension options of RPCs that are issued from TP1/Client/J. To specify multiple extension levels of the RPC facility, specify the disjunction of each value.

Parameter

`extendoption`

Specifies the following extension level of the RPC facility:

`DCRPC_SCD_LOAD_PRIORITY`: Specifies whether or not a TP1/Server that is the gateway that accepts a service request has priority for load balancing. If this option is true, the operation is the same as when `dcscdloadpriority=Y` is specified in the TP1/Client/J environment definition. If this option is false, the operation is the same as when `dcscdloadpriority=N` is specified in the TP1/Client/J environment definition.

`DCRPC_WATCHTIMINHERIT`: Specifies whether or not the RAP-processing server is to inherit the CUP's maximum time to wait for a response when an RPC is issued using the remote API facility. If this option is true, the operation is the same as when `dcwatchtiminherit=Y` is specified in the TP1/Client/J environment definition. If this option is false, the operation is the same as when `dcwatchtiminherit=N` is specified in the TP1/Client/J environment definition.

`DCRPC_RAP_AUTOCONNECT`: Specifies whether or not TP1/Client/J is to establish connection automatically when an RPC is issued using the remote API facility. If this option is true, the operation is the same as when `dcrapautoconnect=Y` is specified in the TP1/Client/J environment definition. If this option is false, the operation is the same as when `dcrapautoconnect=N` is specified in the TP1/Client/J environment definition. This option is ignored if connection has already been established by calling the `openConnection` method in the non-auto connect mode.

`DCRPC_WATCHTIMRPCINHERIT`: Specifies whether or not the server is to inherit the CUP's maximum time to wait for a response. If this option is true, the operation is the same as when `dcwatchtimrpcinherit=Y` is specified in the TP1/Client/J environment definition. If this option is false, the operation is the same as when `dcwatchtimrpcinherit=N` is specified in the TP1/Client/J environment definition.

Return value

None.

Exception

`ErrInvalidArgsException`

The specified `extendoption` argument is invalid.

■ `setDchost`

```
public void setDchost(String host, int port)
               throws ErrHostUndefException,
                      ErrInvalidPortException,
                      ErrProtoException
```

Sets the host name and port of the TP1/Server as a gateway.

Parameters

`host`

Specifies the host name of the TP1/Server as a gateway. If you are issuing RPCs that use the remote API facility, specify the host name of the communication-target RAP-processing listener. If there is a firewall between the CUP and the TP1/Server, specify the host name of the firewall.

`port`

Specifies the port number of the scheduler server running on the TP1/Server as a gateway and the port number of the name server. If you are issuing RPCs that use the remote API facility, specify the port number of the communication-target RAP-processing listener. Express the port number in the range from 5001 to 65535.

Return value

None.

Exceptions

`ErrHostUndefException`

The specified `host` argument is invalid.

`ErrInvalidPortException`

The specified `port` argument is invalid.

`ErrProtoException`

The method issuance order is invalid. This method was called when an RPC was issued using the remote API facility and a permanent connection had already been established with the RAP-processing server.

■ `setTraceArray`
```
public void setTraceArray(String[] array)
```

Specifies whether or not an error trace is to be acquired in the array specified in the argument. You can call this method whether the CUP is a Java applet, application, or servlet.

Parameter

`array`

Specifies the `String` array for storing an error trace. If you specify null, error trace is not acquired.

Return value

None.

■ `setUapTraceMode`
```
public void setUapTraceMode(String TrcPath,
                            int size,
                            boolean flag)
                     throws ErrInvalidArgsException
```

Specifies whether or not a UAP trace is to be acquired when the CUP is a Java application or servlet. If the CUP is a Java applet, do not call this method.

Parameters

`TrcPath`

Specifies the directory in which the UAP trace is to be output. This parameter is ignored if `false` is specified in the `flag` argument.

`size`

Specifies the size of the UAP trace file that is to be output, in the range from 4096 to 1048576. This parameter is ignored if `false` is specified in the `flag` argument.

`flag`

Specifies whether or not a UAP trace is to be acquired:

`true`: Acquire UAP trace.

`false`: Do not acquire UAP trace.

Return value

None.

Exception

`ErrInvalidArgsException`

The specified argument is invalid.

Note

You cannot use this method with a Java applet. If used, operations cannot be guaranteed.

■ `setMethodTraceMode`
```
public void setMethodTraceMode(String TrcPath,
                               int size,
                               boolean flag)
                        throws ErrInvalidArgsException
```

Specifies whether or not a method trace is to be acquired when the CUP is a Java application or servlet. If the CUP is a Java applet, do not call this method.

Parameters

`TrcPath`

Specifies the directory in which the method trace is to be output. This parameter is ignored if `false` is specified in the `flag` argument.

`size`

Specifies the size of the method trace file that is to be output, in the range from 4096 to 1048576. This parameter is ignored if `false` is specified in the `flag` argument.

`flag`

Specifies whether or not a method trace is to be acquired:

`true`: Acquire method trace.

`false`: Do not acquire method trace.

Return value

None.

Exception

`ErrInvalidArgsException`

The specified argument is invalid.

Note

You cannot use this method with a Java applet. If used, operations cannot be guaranteed.

■ `setDataTraceMode`

```
public void setDataTraceMode(String TrcPath,
                             int size,
                             int DataSize,
                             boolean flag)
                      throws ErrInvalidArgsException
```

Specifies whether or not a data trace is to be acquired when the CUP is a Java application or servlet. If the CUP is a Java applet, do not call this method.

Parameters

`TrcPath`

Specifies the directory in which the data trace is to be output. This parameter is ignored if `false` is specified in the `flag` argument.

`size`

Specifies the size of the data trace file that is to be output, in the range from 4096 to 1048576. This parameter is ignored if `false` is specified in the `flag` argument.

`DataSize`

Specifies the size of the data that is to be output, in the range from 16 to 1048576. This parameter is ignored if `false` is specified in the `flag` argument.

`flag`

Specifies whether or not a data trace is to be acquired:

`true`: Acquire data trace.

`false`: Do not acquire data trace.

Return value

None.

Exception

`ErrInvalidArgsException`

The specified argument is invalid.

Note

You cannot use this method with a Java applet. If used, operations cannot be guaranteed.

- `setErrorTraceMode`

```
public void setErrorTraceMode(String TrcPath,
                              int size,
                              boolean flag)
                       throws ErrInvalidArgsException
```

Specifies whether or not an error trace is to be acquired when the CUP is a Java application or servlet. If the CUP is a Java applet, do not call this method.

Parameters

`TrcPath`

Specifies the directory in which the error trace is to be output. This parameter is ignored if `false` is specified in the `flag` argument.

`size`

Specifies the size of the error trace file that is to be output, in the range from 4096 to 1048576. This parameter is ignored if `false` is specified in the `flag` argument.

`flag`

Specifies whether or not an error trace is to be acquired:

`true`: Acquire error trace.

`false`: Do not acquire error trace.

Return value

None.

Exception

`ErrInvalidArgsException`

The specified argument is invalid.

Note

You cannot use this method with a Java applet. If used, operations cannot be guaranteed.

■ `rpcOpen`

```
public void rpcOpen()
             throws ErrIOErrException,
                    ErrProtoException,
                    ErrFatalException,
                    ErrSecurityException,
                    ErrSyserrException
```

Initializes the environment required to call TP1/Server's SPP. To execute a CUP, you must call this method first.

Parameter

None.

Return value

None.

Exceptions

`ErrIOErrException`

An I/O exception occurred.

`ErrProtoException`

The method issuance order was invalid. The `rpcOpen` method was called again before the `rpcClose` method was called.

`ErrFatalException`

There is an error in a TP1/Client/J environment definition specification, or initialization of the environment for communication with TP1/Server failed.

`ErrSyserrException`

A system error occurred.

`ErrSecurityException`

A security exception occurred.

Notes

- To use this method with a Java application or servlet, acquire the TP1/Client/J environment definition from the system properties.
- To use this method with a Java applet, acquire the TP1/Client/J environment definition from the applet's `param` tag.
- To use this method with a Java applet, use `TP1Client(Applet app)` in the constructor that instantiates the `TP1Client` class.

■ `rpcOpen`

```
public void rpcOpen(String deffilename)
             throws ErrIOErrException,
                    ErrProtoException,
                    ErrFatalException,
                    ErrSyserrException,
                    ErrInvalidArgsException,
                    ErrSecurityException
```

Initializes the environment for calling TP1/Server's SPP. To execute a CUP, you must call this method first.

Parameter

`deffilename`

Specifies the complete path name of the file that contains the TP1/Client/J environment definition.

Return value

None.

Exceptions

`ErrIOErrException`

An I/O exception occurred.

`ErrProtoException`

The method issuance order was invalid. The `rpcOpen` method was called again before the `rpcClose` method was called.

`ErrFatalException`

There is an error in the TP1/Client/J environment definition, or initialization of the environment for communication with TP1/Server failed.

`ErrSyserrException`

A system error occurred.

`ErrInvalidArgsException`

A specified argument is invalid.

`ErrSecurityException`

A security exception occurred.

Note

You cannot use this method with a Java applet. When using a Java applet, use the `rpcOpen` method described before this method.

- `rpcClose`

```
public void rpcClose()
              throws ErrIOErrException,
                     ErrSyserrException,
                     ErrNetDownException
```

Releases the environment for calling TP1/Server's SPP. To call the service again for TP1/Server's SPP, call the `rpcOpen` method. Call this method at the end of CUP execution.

Parameter

None.

Return value

None.

Exceptions

`ErrIOErrException`

An I/O exception occurred.

`ErrSyserrException`

A system error occurred.

`ErrNetDownException`

A network error occurred.

- `openConnection`

```
public void openConnection()
                    throws ErrIOErrException,
                           ErrHostUndefException,
                           ErrTimedOutException,
                           ErrNetDownException,
                           ErrNoBufsException,
                           ErrNotUpException,
                           ErrSyserrException,
                                    ErrProtoException,
                                    ErrInvalidArgsException
```

Establishes a permanent connection with the RAP-processing server specified by the `dchost` operand and `dcrapport` operand in the TP1/Client/J environment definition, or connects to the server (TP1/Web) that uses the OpenTP1 Web session management facility specified by the `dcweburl` operand, and starts a virtual session.

You cannot establish multiple concurrent permanent connections from a single CUP.

Note, also, if a firewall is located between the CUP and the RAP-processing listener to which the connection request is directed, you must specify to the target RAP-processing listener the host name and the port number of the firewall.

Parameter

None.

Return value

None. If the method does not return an exception, it has terminated normally or a permanent connection had already been established.

Exceptions

`ErrIOErrException`

An I/O exception occurred.

`ErrHostUndefException`

The host name of a RAP-processing listener is not specified in the `dchost` operand of the TP1/Client/J environment definition.

Another possibility is that the URL (information such as the protocol, Web server, CGI name of the prompter, or TP1/Web service name) specified in the `dcweburl` operand of the TP1/Client/J environment definition is invalid.

`ErrTimedOutException`

A timeout occurred while a connection with a RAP-processing listener was being established.

Another possibility is that a timeout occurred while a virtual session with TP1/Web was being started.

`ErrNetDownException`

A network error occurred during communication with a RAP-processing listener or TP1/Web.

Another possibility is that the communication target TP1/Server is not running.

`ErrNoBufsException`

A memory shortage occurred on the RAP-processing listener or RAP-processing server.

Another possibility is that a memory shortage occurred on TP1/Web.

`ErrNotUpException`

The RAP-processing listener or RAP-processing server is not running.

Another possibility is that TP1/Web is not running.

`ErrSyserrException`

A system error occurred.

`ErrProtoException`

The method issuance order was invalid. Connection had already been established, but the `openConnection` method was called again.

`ErrInvalidArgsException`

A parameter is invalid.

Note

If the connection-target RAP-processing listener is inactive, `ErrIOErrException` or `ErrNetDownException` is returned.

When this method returns an exception, the permanent connection has not been established.

■ `openConnection`
```
public void openConnection(String host,
                           int port)
                    throws ErrIOErrException,
                           ErrHostUndefException,
                           ErrTimedOutException,
                           ErrNoBufsException,
                           ErrNotUpException,
                           ErrSyserrException,
                           ErrInvalidPortException,
                           ErrProtoException,
                           ErrInvalidArgsException,
                           ErrNetDownException
```

Establishes a permanent connection between a CUP and a RAP-processing listener or server in order to issue an RPC that uses the remote API facility. The method determines the target with which permanent connection is to be established from the specified parameter values.

Parameters

`host`

Specifies the host name of the RAP-processing listener or firewall.

`port`

Specifies the port number of the RAP-processing listener or firewall, expressed in the range from 5001 to 65535.

Return value

None. If the method returns no exception, it has terminated normally or a permanent connection had already been established.

Exceptions

`ErrIOErrException`

An I/O exception occurred.

`ErrHostUndefException`

The specified `host` argument is invalid.

`ErrTimedOutException`

A timeout occurred while connection with the RAP-processing listener was being established.

`ErrNoBufsException`

A memory shortage occurred in the RAP-processing listener or server.

`ErrNotUpException`

The RAP-processing listener or server is not running.

`ErrSyserrException`

A system error occurred.

`ErrInvalidPortException`

The specified `port` argument is invalid.

`ErrProtoException`

The method issuance order was invalid. Connection had already been established, but the `openConnection` method was called again.

`ErrInvalidArgsException`

A specified argument is invalid.

`ErrNetDownException`

A network error occurred during communication with the RAP-processing listener, or the communication-target TP1/Server is not active.

Note

If the connection-target RAP-processing listener is inactive, `ErrIOErrException` or `ErrNetDownException` is returned. If this method returns an exception, the permanent connection has not been established.

■ `openConnection`

```
public void openConnection(String url,
                           short flags)
                    throws ErrIOErrException,
                           ErrHostUndefException,
                           ErrTimedOutException,
                           ErrNetDownException,
                           ErrNoBufsException,
                           ErrNotUpException,
                           ErrSyserrException,
                           ErrProtoException,
                           ErrInvalidArgsException
```

Connects to a server (TP1/Web) that uses the OpenTP1 Web session management facility, and starts a virtual session. Subsequently, communication services based on the HTTP protocol can be performed.

Parameters

`url`

Specifies the protocol, Web server, CGI name of the prompter, TP1/Web service name, and other information in the syntax of a URL.

`flags`

Specifies `DCSESSION`.

Return value

None

Exceptions

`ErrIOErrException`

An I/O exception occurred.

`ErrHostUndefException`

The URL (protocol, Web server, CGI name of the prompter, TP1/Web service name, and other information) specified by the `url` parameter is invalid.

`ErrTimedOutException`

A timeout occurred while the virtual session with TP1/Web was starting.

`ErrNetDownException`

A network error occurred.

`ErrNoBufsException`

A memory shortage occurred on TP1/Web.

`ErrNotUpException`

TP1/Web is not running.

`ErrSyserrException`

A system error occurred.

`ErrProtoException`

The issuing sequence of the method is invalid.

`dcrapautoconnect=Y` or `dcrapdirect=N` was specified in the TP1/Client/J environment definition when a permanent connection had already been established with a RAP-processing server.

`ErrInvalidArgsException`

A parameter specification is invalid.

■ `closeConnection`

```
public void closeConnection()
                     throws ErrIOErrException,
                            ErrSyserrException,
                            ErrProtoException,
                            ErrTimedOutException,
                            ErrNetDownException
```

Releases the permanent connection between the CUP and the RAP-processing listener or server.

Parameter

None.

Return value

None.

Exceptions

`ErrIOErrException`

An I/O exception occurred.

`ErrSyserrException`

A system error occurred.

`ErrProtoException`

The method issuance order was invalid. The connection had already been released, but the `closeConnection` method was called again.

`ErrTimedOutException`

A timeout occurred while a virtual session with TP1/Web or a connection with the RAP-processing listener or RAP-processing server was being released.

`ErrNetDownException`

A network error occurred during communication with the RAP-processing listener.

Note

Even if this method returns an exception, the permanent connection has been released.

■ `rpcCall`

```
public void rpcCall(String group,
                    String service,
                    byte[] in_data,
                    int[] in_len,
                    byte[] out_data,
                    int[] out_len,
                    int flags)
             throws ErrInvalidArgsException,
                    ErrProtoException,
                    ErrNoBufsException,
                    ErrNetDownException,
                    ErrTimedOutException,
                    ErrMessageTooBigException,
                    ErrReplyTooBigException,
                    ErrNoSuchServiceGroupException,
                    ErrNoSuchServiceException,
                    ErrServiceClosedException,
                    ErrServiceTerminatingException,
                    ErrServiceNotUpException,
                    ErrNotUpException,
                  ErrSyserrAtServerException,
                  ErrSyserrException,
                  ErrNoBufsAtServerException,
                  ErrInvalidReplyException,
                  ErrInitializingException,
                  ErrTrnchkException,
                  ErrServerBusyException,
                  ErrSecchkException,
                  ErrServiceTerminatedException,
                  ErrIOErrException,
                  ErrTestmodeException,
                  ErrConnfreeException,
                  ErrHostUndefException,
                  ErrInvalidPortException
```

Issues an SPP service request. This method calls the service function that is identified by the service group name and service name and receives its response. If the specified service group is shut down when this method is called, the method returns `ErrServiceClosedException`.

By specifying `DCRPC_TPNOTRAN` in the `flags` argument, you can treat an RPC from transaction processing as a service request that is not transaction processing.

If the specified service group is under termination processing by a command such as `dcsvstop` or has already been terminated when this method is called, `ErrServiceTerminatingException`, `ErrServiceClosedException`, or `ErrNoSuchServiceGroupException` is returned. The actual exception that is returned depends on the timing of this method call.

The server that receives requests from a socket uses the values of the `max_socket_msg` and `max_socket_msglen` operands in the user service definition to execute congestion control on messages. Therefore, the server may not be able to receive service requests. In such a case, calling the method returns `ErrServerBusyException`. When this exception is returned, you may be able to issue a service request by calling this method again after waiting a while.

Parameters

`group`

Specifies the service group name, expressed as an identifier of a maximum of 31 characters.

`service`

Specifies the service name, expressed as an identifier of a maximum of 31 characters.

`in_data`

Specifies input parameters for the service.

`in_len`

Specifies the length of the input parameter for the service, in the range from 1 to the value of `DCRPC_MAX_MESSAGE_SIZE`.[#] Store the length of the input parameter in `in_len[0]`.

#

When the `dccltrpcmaxmsgsize` operand is used, the size of the input parameter response area is the value specified by the `dccltrpcmaxmsgsize` operand rather than the value specified in `DCRPC_MAX_MESSAGE_SIZE`.

`out_data`

Specifies the area in which the response specified in the service function is to be returned.

For a non-response type RPC, specify null. Even if a non-null value is specified for a non-response type RPC, nothing will be stored.

`out_len`

Specifies the length of the service's response, expressed in the range from 1 to the value of `DCRPC_MAX_MESSAGE_SIZE`[#]. Store the length of the response in `out_len[0]`.

When the service request is terminated, the length of the response specified in the service function of the SPP is set in `out_len[0]`.

For a non-response type RPC, the length of response is ignored, if specified, in which case nothing is set in `out_len[0]` when the service request is terminated.

#

When the `dccltrpcmaxmsgsize` operand is used, the length of the service's response is the value specified by the `dccltrpcmaxmsgsize` operand rather than the value specified in `DCRPC_MAX_MESSAGE_SIZE`.

`flags`

Specifies one of the following flags as the RPC mode:

`DCNOFLAGS`: Synchronous-response RPC

`DCRPC_NOREPLY`: Non-response type RPC

`DCRPC_CHAINED`: Chained RPC

`DCRPC_TPNOTRAN`: RPC that does not inherit transactions

If you specify `DCNOFLAGS` or `DCRPC_CHAINED`, the method does not return control until the server returns a response, or the maximum time to wait for a response specified in the `dcwatchtim` operand of the TP1/Client/J environment definition is reached and a timeout occurs. If the SPP to which the service request was issued terminates abnormally, the method returns an exception immediately. In this case, the exception returned by the method depends on the value of the `dcwatchtim` operand specified in the TP1/Client/J environment definition:

- When a value in the range from 1 to 65535 is specified

  `ErrTimedOutException`
- When 0 is specified

  `ErrServiceNotUpException`

If `DCRPC_NOREPLY` is specified, the method assumes that the requested service does not return a response. In such a case, the method returns control immediately without waiting for the service to end its execution. When you specify this flag, you cannot reference the response (value specified in the `out_data` argument) or the length of the response (value specified in the `out_len` argument). Additionally, the CUP cannot determine whether or not the service function was executed.

If `DCRPC_CHAINED` is specified and a service belonging to the same service group is requested more than once, the same process as for the first request is used for execution.

When a chained RPC is used, the following limitations apply:

- The second or subsequent call to this method cannot detect a shutdown of the user server or service.
- If an error occurs in service function processing during the second or subsequent call to this method, the entire user server is shut down, not just the applicable service.
- If you use the remote API facility, you can specify `DCRPC_CHAINED` in the `flags` parameter.
- In the case of a chained RPC, use one of the following methods to terminate it:

  - Execute the `rpcCall` method specifying `DCNOFLAGS` in the `flags` parameter (synchronous-response RPC) on the service group that is executing the chained RPC.#
  - Complete the global transaction executing the chained RPC by

synchronization point processing (commit or rollback).

#

If you call the `closeConnection` or `rpcClose` method without executing a synchronous-response RPC, the API of TP1/Client/J terminates normally, but the following status results:

- Outside the global transaction range

The process executing the service is locked until a chained RPC timeout occurs.

- Within the global transaction range

Implicit commit occurs and the chained RPC is terminated.

By specifying `DCRPC_TPNOTRAN`, you can treat an RPC from transaction processing as a service request that is not transaction processing. You can specify `DCRPC_TPNOTRAN` only from transaction processing. You can specify `DCRPC_TPNOTRAN` with `DCNOFLAGS`, `DCRPC_NOREPLY`, or `DCRPC_CHAINED`.

Example: `flags = TP1Client.DCNOFLAGS|TP1Client.DCRPC_TPNOTRAN;`

Return value

None.

Exceptions

`ErrInvalidArgsException`

A specified argument is invalid. In this case, the invalid argument name is set in the detail message.

`ErrProtoException`

The method issuance order was invalid. The `openConnection` method has not been called.

`ErrNoBufsException`

A memory shortage occurred.

`ErrNetDownException`

A network error occurred, or the communication-target TP1/Server is not active.

`ErrTimedOutException`

A timeout occurred in this method processing, or the SPP whose service was requested terminated abnormally before completing the processing.

`ErrMessageTooBigException`

The length of input parameters specified in the `in_len` argument exceeds the maximum value.

`ErrReplyTooBigException`

The response returned form the server is larger than the area provided by the CUP (value specified for the `out_data` argument).

`ErrNoSuchServiceGroupException`

The service group name specified in the `group` argument is undefined.

`ErrNoSuchServiceException`

The service name specified in the `service` argument is undefined.

`ErrServiceClosedException`

The service group containing the service name specified in the `service` argument is shut down.

`ErrServiceTerminatingException`

The service specified in the `service` argument is under termination processing.

`ErrServiceNotUpException`

The SPP specified in the service request has not started, or the SPP specified in the service request terminated abnormally before completing its processing. This exception is returned when `dcwatchtim=0` is specified (to wait for a response indefinitely) in the TP1/Client/J environment definition.

`ErrNotUpException`

The TP1/Server is not active at the node that contains the specified service. In this case, the TP1/Server may have terminated abnormally, may have been shut down, may be under termination processing, or a network error may have occurred.

`ErrSyserrAtServerException`

A system error occurred in the specified service.

`ErrSyserrException`

A system error occurred.

`ErrNoBufsAtServerException`

A memory shortage occurred in the specified service.

`ErrInvalidReplyException`

The length of the response returned from the service function is not within the range from 1 to the value specified for `DCRPC_MAX_MESSAGE_SIZE`#.

#

When the `dccltrpcmaxmsgsize` operand is used, the value specified in the `dccltrpcmaxmsgsize` operand is used rather than the value specified in `DCRPC_MAX_MESSAGE_SIZE`.

`ErrInitializingException`

The TP1/Server at the node specified in the service request is under start processing.

`ErrTrnchkException`

In the environment using the inter-node load-balancing facility, the transaction attributes of multiple SPPs do not match. Alternatively, the inter-node load-balancing facility cannot be executed because the version of the TP1/Server at the target node is older than the TP1/Client/J version. This exception is returned only when a service request is issued to the SPP using the inter-node load-balancing facility.

`ErrServerBusyException`

The target server that receives requests from a socket cannot receive service requests.

`ErrSecchkException`

The SPP to which the service request was issued is protected by the security facility. The CUP that called the `rpcCall` method does not have access authority to the SSP.

`ErrServiceTerminatedException`

The SPP specified in the service request terminated abnormally before completing the processing. This exception is returned only when `00000001` is specified in the `rpc_extend_function` operand of the RAP-processing listener service definition. If you specified `00000000` in the `rpc_extend_function` operand or omitted the operand, `ErrTimedOutException` or `ErrServiceNotUpException` is returned.

`ErrIOErrException`

An I/O exception occurred. The RAP-processing server may have timed out while monitoring the inquiry interval and released the connection.

`ErrTestmodeException`

A service request was issued to an SPP in the test mode.

`ErrConnfreeException`

The permanent connection with the RAP-processing server was released. Another possibility is that the virtual session with TP1/Web ended.

`ErrHostUndefException`

Possible causes are as follows:

- Either the host name of the communication target TP1/Server is not specified in the `dchost` operand of the TP1/Client/J environment definition, or the specification is invalid.
- The URL (protocol, Web server, CGI name of the prompter, TP1/Web service name, and other information) specified in the `dcweburl` operand of the TP1/Client environment definition is invalid.

`ErrInvalidPortException`

Possible causes are as follows:

- An RPC was issued using the remote API facility, but the `dcrapport` operand was not specified in the TP1/Client/J environment definition.
- An RPC was issued using the scheduler direct facility, but the `dcscdport` operand was not specified in the TP1/Client/J environment definition.

Notes

- Do not specify the same area for the input parameters (value of the `in_data` argument) and for the service function's response (value of the `out_data` argument).
- If you specify `DCRPC_NOREPLY` in the `flags` argument, the following exceptions will not be returned:

Exceptions that will not occur:

- `ErrReplyTooBigException`
- `ErrInvalidReplyException`

Exceptions that cannot be detected, if thrown:

- `ErrNoSuchServiceException`
- `ErrServiceClosedException`
- `ErrServiceTerminatingException`
- `ErrSyserrAtServerException`
- `ErrNoBufsAtServerException`

    - `ErrNotUpException`

- The possible causes of `ErrTimedOutException` are as follows:

  - The maximum time to wait for a response specified in the TP1/Client/J environment definition or TP1/Server definition is too short.

  - The service function issued by the target SPP terminated abnormally.

  - An error occurred at the node that contains the target SPP.

  - The target SPP terminated abnormally before completing its processing.

  - A network error occurred.

  If any of these events occurs, the transaction started by the target SPP may have been committed and the database may have been updated. Check to see if the database has been updated.

- If an error occurs when this method is called, an exception is returned, but whether or not the permanent connection with the RAP-processing listener or server was released as a result of the error cannot be determined at this point. If the permanent connection with the RAP-processing listener or server has in fact been released, calling this method again returns `ErrConnfreeException`.

- If the permanent connection with the RAP-processing listener or server is released as a result of an error when this method is called, you need to call the `openConnection` method again to establish permanent connection with the RAP-processing listener or server.

- In the case of a permanent connection while the auto connect mode is in effect, if a network error occurs in communication with the RAP-processing server during transmission of the service request, the method retries only once. If the connection is not established by this retry, the method returns `ErrNetDownException`.

- In the case of an RPC that uses the scheduler direct facility or name service, you cannot specify `DCRPC_CHAINED` in the `flags` argument. If you use the remote API facility, you can specify `DCRPC_CHAINED` in the `flags` parameter.

■ `rpcCall`

```
public void rpcCall(String group,
                    String service,
                    byte[] in_data,
                    byte[] out_data,
                    int flags)
             throws ErrInvalidArgsException,
                    ErrProtoException,
                    ErrNoBufsException,
                        ErrNetDownException,
                        ErrTimedOutException,
                        ErrMessageTooBigException,
                        ErrReplyTooBigException,
                        ErrNoSuchServiceGroupException,
                        ErrNoSuchServiceException,
                        ErrServiceClosedException,
                        ErrServiceTerminatingException,
                        ErrServiceNotUpException,
                        ErrNotUpException,
                        ErrSyserrAtServerException,
                        ErrSyserrException,
                        ErrNoBufsAtServerException,
                        ErrInvalidReplyException,
                        ErrInitializingException,
                        ErrTrnchkException,
                        ErrServerBusyException,
                        ErrSecchkException,
                        ErrServiceTerminatedException,
                        ErrIOErrException,
                        ErrTestmodeException,
                        ErrConnfreeException,
                        ErrHostUndefException,
                        ErrInvalidPortException
```

Issues an SPP service request. This method calls the service function that is identified by the service group name and service name and receives its response. If the specified service group is shut down when this method is called, the method returns `ErrServiceClosedException`.

By specifying `DCRPC_TPNOTRAN` in the `flags` argument, you can treat an RPC from transaction processing as a service request that is not transaction processing.

If the specified service group is under termination processing by a command such as `dcsvstop` or has already been terminated when this method is called, `ErrServiceTerminatingException`, `ErrServiceClosedException`, or `ErrNoSuchServiceGroupException` is returned. The actual exception that is returned depends on the timing of this method call.

The server that receives requests from a socket uses the values of the `max_socket_msg` and `max_socket_msglen` operands in the user service definition to execute congestion control on messages. Therefore, the server may not be able to receive service requests. In such a case, calling the method returns `ErrServerBusyException`. If this exception is returned, you may be able to issue a service request by calling this method again after waiting a while.

Parameters

`group`

Specifies the service group name, expressed as an identifier of a maximum of 31 characters.

`service`

Specifies the service name, expressed as an identifier of a maximum of 31 characters.

`in_data`

Specifies input parameters for the service.

`out_data`

Specifies the area in which the response specified in the service function is to be returned.

For a non-response type RPC, specify null. Even if a non-null value is specified for a non-response type RPC, nothing will be stored.

`flags`

Specifies one of the following flags as the RPC mode:

`DCNOFLAGS`: Synchronous-response RPC

`DCRPC_NOREPLY`: Non-response type RPC

`DCRPC_CHAINED`: Chained RPC

`DCRPC_TPNOTRAN`: RPC that does not inherit transactions

If you specify `DCNOFLAGS` or `DCRPC_CHAINED`, the method does not return control until the server returns a response, or the maximum time to wait for a response specified in the `dcwatchtim` operand of the TP1/Client/J environment definition is reached and a timeout occurs. If the SPP to which the service request was issued terminates abnormally, the method returns an exception immediately. In this case, the exception returned by the method depends on the maximum time to wait for a response that was specified in the `dcwatchtim` operand of the TP1/Client/J environment definition:

- When a value in the range from 1 to 65535 is specified

  `ErrTimedOutException`
- When 0 is specified

  `ErrServiceNotUpException`

If `DCRPC_NOREPLY` is specified, the method assumes that the requested service does not return a response. In such a case, the method returns control immediately without waiting for the service to end its execution. When you specify this flag, you cannot reference the response (value specified in the `out_data` argument) or the response length (value specified in the `out_len` argument). Additionally, the CUP cannot determine whether or not the service function was executed.

If `DCRPC_CHAINED` is specified and a service belonging to the same service group is requested more than once, the same process as for the first request is used for execution.

When a chained RPC is used, the following limitations apply:

- The second or subsequent call to this method cannot detect a shutdown of the user server or service.
- If an error occurs in service function processing during the second or subsequent call to this method, the entire user server is shut down, not just the applicable service.
- If you specified `DCRPC_CHAINED` in the `flags` argument to call a service, execute an RPC specifying `DCNOFLAGS` during the last service request. If you call the `closeConnection` method without specifying `DCNOFLAGS` during the last service request, the method returns `ErrTimedOutException` or some other exception. The process executing the service is locked until a timeout occurs in the chained RPCs.
- If you use the remote API facility, you can specify `DCRPC_CHAINED` in the `flags` parameter.
- In the case of a chained RPC, use one of the following methods to terminate it:

  - Execute the `rpcCall` method specifying `DCNOFLAGS` in the `flags` parameter (synchronous-response RPC) on the service group that is executing the chained RPC.#
  - Complete the global transaction executing the chained RPC by synchronization point processing (commit or rollback).

  #

  If you call the `closeConnection` or `rpcClose` method without executing a synchronous-response RPC, the API of TP1/Client/J terminates normally, but the following status results:

  - Outside the global transaction range

    The process executing the service is locked until a chained RPC timeout occurs.

  - Within the global transaction range

    Implicit commit occurs and the chained RPC is terminated.

By specifying `DCRPC_TPNOTRAN`, you can treat an RPC from transaction processing as a service request that is not transaction processing. You cannot specify `DCRPC_TPNOTRAN` from processing outside the transaction. You can specify `DCRPC_TPNOTRAN` with `DCNOFLAGS`, `DCRPC_NOREPLY`, or `DCRPC_CHAINED`.

Example: `flags = TP1Client.DCNOFLAGS|TP1Client.DCRPC_TPNOTRAN;`

Return value

None.

Exceptions

`ErrInvalidArgsException`

A specified argument is invalid. In this case, the invalid argument name is set in the detail message.

`ErrProtoException`

The method issuance order was invalid. The `openConnection` method has not been called.

`ErrNoBufsException`

A memory shortage occurred.

`ErrNetDownException`

A network error occurred, or the communication-target TP1/Server is not active.

`ErrTimedOutException`

A timeout occurred in this method processing, or the SPP whose service was requested terminated abnormally before completing the processing.

`ErrMessageTooBigException`

The input parameters length specified in the `in_len` argument exceeds the maximum value.

`ErrReplyTooBigException`

The response returned form the server is larger than the area provided by the CUP (value specified for the `out_data` argument).

`ErrNoSuchServiceGroupException`

The service group name specified in the `group` argument is undefined.

`ErrNoSuchServiceException`

The service name specified in the `service` argument is undefined.

`ErrServiceClosedException`

The service group containing the service name specified in the `service` argument is shut down.

`ErrServiceTerminatingException`

The service specified in the `service` argument is under termination processing.

`ErrServiceNotUpException`

The SPP specified in the service request has not started, or the SPP specified in the service request terminated abnormally before completing its processing. This exception is returned when `dcwatchtim=0` is specified (to wait for a response indefinitely) in the TP1/Client/J environment definition.

`ErrNotUpException`

The TP1/Server is not active at the node that contains the specified service. In this case, the TP1/Server may have terminated abnormally, may have been shut down, may be under termination processing, or a network error may have occurred.

`ErrSyserrAtServerException`

A system error occurred in the specified service.

`ErrSyserrException`

A system error occurred.

`ErrNoBufsAtServerException`

A memory shortage occurred in the specified service.

`ErrInvalidReplyException`

The length of the response returned from the service function is not within the range from 1 to the value specified for `DCRPC_MAX_MESSAGE_SIZE`#.

#

When the `dccltrpcmaxmsgsize` operand is used, the value specified in the `dccltrpcmaxmsgsize` operand is used rather than the value specified in `DCRPC_MAX_MESSAGE_SIZE`.

`ErrInitializingException`

The TP1/Server at the node specified in the service request is under start processing.

`ErrTrnchkException`

In the environment using the inter-node load-balancing facility, the transaction attributes of multiple SPPs do not match, or the inter-node load-balancing facility cannot be executed because the version of the TP1/Server at the target node is older than the TP1/Client/J version. This exception is returned only when a service request is issued to the SPP using the inter-node load-balancing facility.

`ErrServerBusyException`

The target server that receives requests from a socket cannot receive service requests.

`ErrSecchkException`

The SPP to which the service request was issued is protected by the security facility. The CUP that called the `rpcCall` method does not have access authority to the SSP.

`ErrServiceTerminatedException`

The SPP specified in the service request terminated abnormally before completing the processing. This exception is returned only when `00000001` is specified in the `rpc_extend_function` operand of the RAP-processing listener service definition. If you specified `00000000` in the `rpc_extend_function` operand or omitted the operand, `ErrTimedOutException` or `ErrServiceNotUpException` is returned.

`ErrIOErrException`

An I/O exception occurred. The RAP-processing server may have timed out while monitoring the inquiry interval and released the connection.

`ErrTestmodeException`

A service request was issued to an SPP in the test mode.

`ErrConnfreeException`

The permanent connection with the RAP-processing server was released. Another possibility is that the virtual session with TP1/Web ended.

`ErrHostUndefException`

Possible causes are as follows:

- Either the host name of the communication target TP1/Server is not specified in the `dchost` operand of the TP1/Client/J environment definition, or the specification is invalid.
- The URL (protocol, Web server, CGI name of the prompter, TP1/Web service name, and other information) specified in the `dcweburl`

operand of the TP1/Client/J environment definition is invalid.

`ErrInvalidPortException`

Possible causes are as follows:

- An RPC was issued using the remote API facility, but the `dcrapport` operand was not specified in the TP1/Client/J environment definition.
- An RPC was issued using the scheduler direct facility, but the `dcscdport` operand was not specified in the TP1/Client/J environment definition.

Notes

- Do not specify the same area for the input parameters (value of the `in_data` argument) and for the response of the service function (value of the `out_data` argument).
- If you specify `DCRPC_NOREPLY` in the `flags` argument, the following exceptions will not be returned:

Exceptions that will not occur:

- `ErrReplyTooBigException`
- `ErrInvalidReplyException`

Exceptions that cannot be detected, if thrown:

- `ErrNoSuchServiceException`
- `ErrServiceClosedException`
- `ErrServiceTerminatingException`
- `ErrSyserrAtServerException`
- `ErrNoBufsAtServerException`
- `ErrNotUpException`

- The possible causes of `ErrTimedOutException` are as follows:

  - The maximum time to wait for a response specified in the TP1/Client/J environment definition or TP1/Server definition is too short.

  - The service function issued by the target SPP terminated abnormally.

  - An error occurred at the node that contains the target SPP.

  - The target SPP terminated abnormally before completing its processing.

  - A network error occurred.

  If any of these events occurs, the transaction started by the target SPP may have been committed and the database may have been updated. Check to see whether or not the database has been updated.

- If an error occurs when this method is called, an exception is returned, but whether or not the permanent connection with the RAP-processing listener or server was released as a result of the error cannot be determined at this point. If the permanent connection with the RAP-processing listener or server has in fact been released, calling this method again returns `ErrConnfreeException`.
- If the permanent connection with the RAP-processing listener or server is released as a result of an error when this method is called, you need to call the `openConnection` method again to establish permanent connection with the RAP-processing listener or server.
- In the case of a permanent connection when the auto connect mode is being used, if a network error occurs in communication with the RAP-processing server during transmission of the service request, the method retries only once. If the connection is not established by this retry, the method returns `ErrNetDownException`.
- In the case of an RPC that uses the scheduler direct facility or name service, you cannot specify `DCRPC_CHAINED` in the `flags` argument. If you use the remote API facility, you can specify `DCRPC_CHAINED` in the `flags` parameter.

■ `rpcCallTo`

```
public void rpcCallTo(DCRpcBindTbl direction,
                      String group,
                      String service,
                      byte[] in_data,
                      int[] in_len,
                      byte[] out_data,
                      int[] out_len,
                      int flags)
               throws ErrInvalidArgsException,
                      ErrProtoException,
                      ErrNoBufsException,
                      ErrNetDownException,
                      ErrTimedOutException,
                      ErrMessageTooBigException,
                      ErrReplyTooBigException,
                      ErrNoSuchServiceGroupException,
                      ErrNoSuchServiceException,
                      ErrServiceClosedException,
                      ErrServiceTerminatingException,
                      ErrServiceNotUpException,
                      ErrNotUpException,
                    ErrSyserrAtServerException,
                    ErrSyserrException,
                    ErrNoBufsAtServerException,
                    ErrInvalidReplyException,
                    ErrInitializingException,
                    ErrTrnchkException,
                    ErrServerBusyException,
                    ErrSecchkException,
                    ErrServiceTerminatedException,
                    ErrIOErrException,
                    ErrTestmodeException,
                    ErrConnfreeException,
                    ErrHostUndefException,
                    ErrInvalidPortException
```

Similarly to the `rpcCall` method, this method issues an SPP service request. The `rpcCallTo` method uses a host name in addition to a service group name and service name as the search key for determining the target service function.

Before calling this method, you need to call the `setScdDirectObject` method and create a `DCRpcBindTbl` instance. The other interfaces are the same as for the `rpcCall` method.

Parameters

`direction`

Specifies the `DCRpcBindTbl` object. Before calling this method, call the `setScdDirectObject` method to set the communication-target information.

`group`

Specifies the service group name, expressed as an identifier of a maximum of 31 characters.

`service`

Specifies the service name, expressed as an identifier of a maximum of 31 characters.

`in_data`

Specifies input parameters for the service.

`in_len`

Specifies the length of the input parameter for the service, in the range from 1 to the value of `DCRPC_MAX_MESSAGE_SIZE`.# Store the length of the input parameter in `in_len[0]`.

#

When the dccltrpcmaxmsgsize operand is used, the size of the input parameter response area is the value specified by the dccltrpcmaxmsgsize operand rather than the value specified in DCRPC_MAX_MESSAGE_SIZE.

out_data

Specifies the area in which the response specified in the service function is to be returned.

For a non-response type RPC, specify null. Even if a non-null value is specified for a non-response type RPC, nothing will be stored.

out_len

Specifies the length of the service's response, expressed in the range from 1 to the value of DCRPC_MAX_MESSAGE_SIZE#. Store the length of the response in out_len[0].

#

When the dccltrpcmaxmsgsize operand is used, the length of the service's response is the value specified by the dccltrpcmaxmsgsize operand rather than the value specified in DCRPC_MAX_MESSAGE_SIZE.

flags

Specifies one of the following flags as the RPC mode:

DCNOFLAGS: Synchronous-response RPC

DCRPC_NOREPLY: Non-response type RPC

Return value

None.

Exception

Exceptions are the same as for the rpcCall method.

Notes

- If you issue a service request to a user server that receives requests from a socket, this method returns ErrServiceNotUpException.
- The version of the target TP1/Server whose service is requested must be 03-03 or later. If a service request is issued to an earlier version of TP1/ Server, operations cannot be guaranteed.
- This method is not applicable to an RPC that uses the remote API facility. If this method is issued when the remote API facility is used, the method

returns `ErrProtoException`.

- This method is not applicable to a chained RPC. If you specify `DCRPC_CHAINED` in the `flags` argument, this method returns `ErrInvalidArgsException`.
- During a call to the `rpcCallTo` method, the following operands in the TP1/Client/J environment definition are not referenced:

  - `dchost`

  - `dcscdport`

  - `dcnamport`

  - `dcscdloadpriority`

  Neither the host name nor the port number specified in the `setDchost` method is referenced. The specified values take effect the next time the `rpcCall` method is called.
- If the specified target host name is invalid, this method returns `ErrHostUndefException`.
- If the specified target port number is invalid, this method returns `ErrInvalidPortException`.

■ `setScdDirectObject`

```
public DCRpcBindTbl setScdDirectObject(String scdhost,
                                       int scdport,
                                       int flags)
```

Sets the host name and port number of the communication-target scheduler.

Parameters

`scdhost`

Specifies the host name of the communication-target schedule server. If the specified host name is invalid or `NULL` is specified, the `rpcCallTo` method that is called after this method returns `ErrHostUndefException`.

`scdport`

Specifies the port number of the communication-target schedule server. Express the port number in the range from 5001 to 65535. If the specified port number is invalid, the `rpcCallTo` method that is called after this method returns `ErrInvalidPortException`.

`flags`

Specifies `DCNOFLAGS`.

Return value

`DCRpcBindTbl` object

■ `trnBegin`

```
public void trnBegin()
             throws ErrProtoException,
                    ErrTMException,
                    ErrRMException,
                    ErrNoBufsException,
                    ErrNotUpException,
                    ErrTimedOutException,
                    ErrIOErrException,
                    ErrConnfreeException,
                    ErrSyserrException,
                    ErrNetDownException,
                    ErrHostUndefException,
                    ErrInvalidPortException
```

Starts a global transaction from the `TP1Client` object that calls the `trnBegin` method.

This method is applicable only when the remote API facility is used and the version of the target TP1/Server Base is 05-00 or later.

Parameter

None.

Return value

None.

Exceptions

`ErrProtoException`

The `trnBegin` method was called from an invalid context.

`ErrTMException`

The method was unable to start the transaction because an error occurred in the transaction service. When this exception is returned, calling this method again after waiting a while may start the transaction successfully.

`ErrRMException`

An error occurred in the resource manager. The transaction was not started.

`ErrNoBufsException`

A memory shortage occurred.

`ErrNotUpException`

TP1/Server is not active.

`ErrTimedOutException`

A timeout occurred.

`ErrIOErrException`

An I/O exception occurred.

`ErrConnfreeException`

The permanent connection was released from the CUP executing process.

`ErrSyserrException`

A system error occurred.

`ErrNetDownException`

A network error occurred.

`ErrHostUndefException`

Either the host name of the communication target TP1/Server is not specified in the `dchost` operand of the TP1/Client/J environment definition, or the specification is invalid.

`ErrInvalidPortException`

The communication-target port is invalid or has not been set.

■ `trnChainedCommit`

```
public void trnChainedCommit()
             throws ErrProtoException,
                    ErrRollbackException,
                    ErrHeuristicException,
                    ErrHazardException,
                    ErrNoBeginException,
                    ErrRollbackNoBeginException,
                    ErrHeuristicNoBeginException,
                    ErrHazardNoBeginException,
                    ErrNoBufsException,
                    ErrNotUpException,
                    ErrTimedOutException,
                    ErrIOErrException,
                    ErrConnfreeException,
                    ErrNetDownException,
                    ErrSyserrException
```

Acquires a transaction synchronization point.

When the `trnChainedCommit` method terminates normally, a new global transaction occurs and the subsequent methods that are executed belong to the range of the new global transaction.

Parameter

None.

Return value

None.

Exceptions

`ErrProtoException`

The `trnChainedCommit` method was called from an invalid context.

`ErrRollbackException`

The current transaction was rolled back without being committed. Even after this exception is returned, this process is still under the transaction and within the range of the global transaction.

`ErrHeuristicException`

For heuristic determination purposes, some of the transaction branches in the global transaction that executed the `dc_trn_chained_commit` function were committed and some were rolled back. This exception is returned when the result of heuristic determination does not match the result of the synchronization point of the global transaction. For the result of the synchronization point of the global transaction, resource manager, and UAP that caused this exception, see TP1/Server's message log file. Even after this exception is returned, this process is still under the transaction and within the range of the global transaction.

`ErrHazardException`

A transaction branch of the global transaction was completed heuristically. However, the result of the synchronization point of the transaction branch that was completed heuristically was unknown due to an error. For the result of the synchronization point of the global transaction, resource manager, and UAP that caused this exception, see TP1/Server's message log file. Even after this exception is returned, this process is still under the transaction and within the range of the global transaction.

`ErrNoBeginException`

Commit or rollback processing terminated normally, but the method was unable to start a new transaction. Once this exception has been returned, this process is no longer under the transaction.

`ErrRollbackNoBeginException`

The transaction that was to be committed was rolled back without being committed. The method was unable to start a new transaction. Once this exception has been returned, this process is no longer under the transaction.

`ErrHeuristicNoBeginException`

For heuristic determination purposes, some of the transaction branches in the global transaction that executed the `dc_trn_chained_commit` function were committed and some were rolled back. This exception is returned when the result of the heuristic determination does not match the result of the synchronization point of the global transaction. For the result of the synchronization point of the global transaction, resource manager, and UAP that caused this exception, see TP1/Server's message log file. The method was unable to start a new transaction. Once this exception has been returned, this process is no longer under the transaction.

`ErrHazardNoBeginException`

A transaction branch of the global transaction was completed heuristically. However, the result of the synchronization point of the transaction branch that was completed heuristically was unknown due to an error. For the result of the synchronization point of the global transaction, resource manager, and UAP that caused this exception, see TP1/Server's message log file. The method was unable to start a new transaction. Once this exception has been returned, this process is no longer under the transaction.

`ErrNoBufsException`

A memory shortage occurred.

`ErrNotUpException`

TP1/Server is not active.

`ErrTimedOutException`

A timeout occurred.

`ErrIOErrException`

An I/O exception occurred.

`ErrConnfreeException`

The permanent connection was released from the CUP executing process.

`ErrNetDownException`

A network error occurred.

`ErrSyserrException`

A system error occurred.

■ `trnChainedRollback`

```
public void trnChainedRollback()
             throws ErrProtoException,
                    ErrHeuristicException,
                    ErrHazardException,
                    ErrNoBeginException,
                    ErrHeuristicNoBeginException,
                    ErrHazardNoBeginException,
                    ErrNoBufsException,
                    ErrNotUpException,
                    ErrTimedOutException,
                    ErrIOErrException,
                    ErrConnfreeException,
                    ErrNetDownException,
                    ErrSyserrException
```

Rolls back a transaction.

When the `trnChainedRollback` method terminates normally, a new global transaction occurs and the subsequent methods that are called belong to the range of the new global transaction.

Parameter

None.

Return value

None.

Exceptions

`ErrProtoException`

The `trnChainedRollback` method was called from an invalid context.

`ErrHeuristicException`

For heuristic determination purposes, some of the transaction branches in the global transaction that executed the `dc_trn_chained_rollback` function were committed and some were rolled back. This exception is returned when the result of the heuristic determination does not match the result of the synchronization point of the global transaction. For the result of the synchronization point of the global transaction, resource manager, and UAP that caused this exception, see TP1/Server's message log file. Even after this exception is returned, this process is still under the transaction and within the range of the global transaction.

`ErrHazardException`

A transaction branch of the global transaction was completed heuristically.

However, the result of the synchronization point of the transaction branch that was completed heuristically was unknown due to an error. For the result of the synchronization point of the global transaction, resource manager, and UAP that caused this exception, see TP1/Server's message log file. Even after this exception is returned, this process is still under the transaction and within the range of the global transaction.

`ErrNoBeginException`

Commit or rollback processing terminated normally, but the method was unable to start a new transaction. Once this exception has been returned, this process is no longer under the transaction.

`ErrHeuristicNoBeginException`

For heuristic determination purposes, some of the transaction branches in the global transaction that executed the `dc_trn_chained_rollback` function were committed and some were rolled back. This exception is returned when the result of the heuristic determination does not match the result of the synchronization point of the global transaction. For the result of the synchronization point of the global transaction, resource manager, and UAP that caused this exception, see TP1/Server's message log file. The method was unable to start a new transaction. Once this exception has been returned, this process is no longer under the transaction.

`ErrHazardNoBeginException`

A transaction branch of the global transaction was completed heuristically. However, the result of the synchronization point of the transaction branch that was completed heuristically was unknown due to an error. For the result of the synchronization point of the global transaction, resource manager, and UAP that caused this exception, see TP1/Server's message log file. The method was unable to start a new transaction. Once this exception has been returned, this process is no longer under the transaction.

`ErrNoBufsException`

A memory shortage occurred.

`ErrNotUpException`

TP1/Server is not active.

`ErrTimedOutException`

A timeout occurred.

`ErrIOErrException`

An I/O exception occurred.

`ErrConnfreeException`

The permanent connection was released from the CUP executing process.

`ErrNetDownException`

A network error occurred.

`ErrSyserrException`

A system error occurred.

■ `trnUnchainedCommit`

```
public void trnUnchainedCommit()
             throws ErrProtoException,
                    ErrRollbackException,
                    ErrHeuristicException,
                    ErrHazardException,
                    ErrNoBufsException,
                    ErrNotUpException,
                    ErrTimedOutException,
                    ErrIOErrException,
                    ErrConnfreeException,
                    ErrNetDownException,
                    ErrSyserrException
```

Acquires a transaction synchronization point.

When the `trnUnchainedCommit` method terminates normally, the global transaction ends. You cannot execute an SPP as a transaction from outside the range of the global transaction.

Parameter

None.

Return value

None.

Exceptions

`ErrProtoException`

The `trnUnchainedCommit` method was called from an invalid context.

`ErrRollbackException`

The current transaction was rolled back without being committed. The process is outside the range of the transaction.

`ErrHeuristicException`

For heuristic determination purposes, some of the transaction branches in the global transaction that executed the `dc_trn_unchained_commit` function were committed and some were rolled back. This exception is returned when the result of the heuristic determination does not match the result of the synchronization point of the global transaction. For the result of the synchronization point of the global transaction, resource manager, and UAP that caused this exception, see TP1/Server's message log file. Once this exception has been returned, this process is no longer under the transaction. The process is outside the range of the global transaction.

`ErrHazardException`

A transaction branch of the global transaction was completed heuristically. However, the result of the synchronization point of the transaction branch that was completed heuristically was unknown due to an error. For the result of the synchronization point of the global transaction, resource manager, and UAP that caused this exception, see TP1/Server's message log file. Once this exception has been returned, this process is no longer under the transaction. The process is outside the range of the global transaction.

`ErrNoBufsException`

A memory shortage occurred.

`ErrNotUpException`

TP1/Server is not active.

`ErrTimedOutException`

A timeout occurred.

`ErrIOErrException`

An I/O exception occurred.

`ErrConnfreeException`

The permanent connection was released from the CUP executing process.

`ErrNetDownException`

A network error occurred.

`ErrSyserrException`

A system error occurred.

■ `trnUnchainedRollback`

```
public void trnUnchainedRollback()
             throws ErrProtoException,
                    ErrHeuristicException,
                    ErrHazardException,
                    ErrNoBufsException,
                    ErrNotUpException,
                    ErrTimedOutException,
                    ErrIOErrException,
                    ErrConnfreeException,
                    ErrNetDownException,
                    ErrSyserrException
```

Rolls back a transaction.

When the `trnUnchainedRollback` method is called, the global transaction ends. You cannot execute an SPP as a transaction from outside the range of the global transaction.

Parameter

None.

Return value

None.

Exceptions

`ErrProtoException`

The `trnUnchainedRollback` method was called from an invalid context.

`ErrHeuristicException`

For heuristic determination purposes, some of the transaction branches in the global transaction that executed the `dc_trn_unchained_rollback` function were committed and some were rolled back. This exception is returned when the result of the heuristic determination does not match the result of the synchronization point of the global transaction. For the result of the synchronization point of the global transaction, resource manager, and UAP that caused this exception, see TP1/Server's message log file. Once this exception has been returned, this process is no longer under the transaction. The process is outside the range of the global transaction.

`ErrHazardException`

A transaction branch of the global transaction was completed heuristically. However, the result of the synchronization point of the transaction branch that was completed heuristically was unknown due to an error. For the result of the synchronization point of the global transaction, resource manager, and UAP that caused this exception, see TP1/Server's message log file. Once this exception has been returned, this process is no longer under the transaction. The process is outside the range of the global transaction.

`ErrNoBufsException`

A memory shortage occurred.

`ErrNotUpException`

TP1/Server is not active.

`ErrTimedOutException`

A timeout occurred.

`ErrIOErrException`

An I/O exception occurred.

`ErrConnfreeException`

The permanent connection was released from the CUP executing process.

`ErrNetDownException`

A network error occurred.

`ErrSyserrException`

A system error occurred.

■ `trnInfo`

```
public boolean trnInfo()
```

Reports whether or not the `TP1Client` object that called the `trnInfo` method is currently running as a transaction. This method only references internal variables; it does not communicate with the RAP-processing server.

Parameter

None.

Return value

`true`

The `TP1Client` object that called the `trnInfo` method is within the range of a transaction.

`false`

The `TP1Client` object that called the `trnInfo` method is outside the range of a transaction.

Exception

None.

■ `getTrnID`

```
public void getTrnID(byte[] gid,byte[] bid)
             throws ErrInvalidArgsException,
                    ErrProtoException
```

Acquires the current transaction global identifier and transaction branch identifier. This method only references internal variables; it does not communicate with the RAP-processing server.

The current transaction global identifier and transaction branch identifier were assigned by TP1/Server when the following methods were called and a transaction was started:

- `trnBegin` method
- `trnChainedCommit` method
- `trnChainedRollback` method

Parameter

`gid`

Specifies a minimum of 16 bytes of the `byte` type array that stores the transaction global identifier.

`bid`

Specifies a minimum of 16 bytes of the `byte` type array that stores the transaction branch identifier.

Return value

None.

Exceptions

`ErrInvalidArgsException`

A specified argument is invalid.

`ErrProtoException`

The `getTrnID` method was called from an invalid context.

■ `cltReceive`

```
public void cltReceive(byte[]  buff,
                       int[]   recvleng,
                       int     timeout,
                       int     flags)
                throws ErrInvalidArgsException,
                       ErrProtoException,
                       ErrNetDownException,
                       ErrTimedOutException,
                       ErrSyserrException,
                       ErrInvalidPortException,
                       ErrConnfreeException
```

Receives a message sent from an MHP.

Before executing the `cltReceive` method, you must have specified either `DCCLT_ONEWAY_RCV` or `DCCLT_SNDRCV` in the `dcsndrcvtype` operand of the TP1/ Client/J environment definition, and executed the `rpcOpen` method.

Parameters

`buff`

Specifies the area in which to store the received message. Make sure to specify an area that is at least the size specified in `recvleng`. The received message is returned after the method finishes executing.

`recvleng`

Specifies the length of the message to be received in `recvleng[0]`. The length of the received message is returned after the method finishes executing.

`timeout`

Specifies the maximum wait time when a message is being received, expressed as an integer from -1 to 65535 in units of seconds.

If you specify `-1`, `cltReceive` waits until the message is received without any timeout limitations.

If you specify `0`, `cltReceive` does not wait for the message to be received. If there is no message to be received, it returns an `ErrTimedOutException`.

If you specify an integer between 1 and 65535, `cltReceive` waits for the specified number of seconds for the message to be received. If the specified number of seconds is exceeded before the message can be received, `cltReceive` returns an `ErrTimedOutException`.

`flags`

Specifies whether or not to release the connection after the message is received.

`DCNOFLAGS`: Does not release the connection after the message is received.

If `DCNOFLAGS` is specified, unless there is an error, the connection is not released until the `rpcClose` method is executed.

`DCCLT_RCV_CLOSE`: Releases the connection after the message is received.

Return value

None

Exceptions

`ErrInvalidArgsException`

The parameter specification is invalid.

`ErrProtoException`

The `rpcOpen` method did not execute. Another possibility is that, although the `rpcOpen` method did execute, neither `DCCLT_ONEWAY_RCV` nor `DCCLT_SNDRCV` is specified in the `dcsndrcvtype` operand of the TP1/Client/J environment definition.

`ErrNetDownException`

A network error occurred.

`ErrTimedOutException`

A timeout occurred while the message was being received.

`ErrSyserrException`

A system error occurred.

`ErrInvalidPortException`

This exception occurs when `Y` is specified in the `dcsockopenatrcv` operand of the TP1/Client/J environment definition, and the port specified in the `dcrcvport` operand is already in use. If `N` is specified in the `dcsockopenatrcv` operand, the `rpcOpen` method returns `ErrFatalException`.

`ErrConnfreeException`

The connection was released from the MHP.

Notes

The `cltReceive` method returns control to the CUP at the following times:

- When a message of the length specified by the `recvleng` parameter is received from the MHP
- When a timeout occurs while a message is being received from the MHP
- When the connection is released from the MHP
- When a network error occurs

If the MHP drops the connection when a `cltReceive` method is issued, an `ErrConnfreeException` error is returned.

■ `cltSend`

```
public void cltSend(byte[]  buff,
                    int     sendleng,
                    String  hostname,
                    int     portnum,
                    int     flags)
             throws ErrInvalidArgsException,
                    ErrProtoException,
                    ErrNetDownException,
                    ErrSyserrException,
                    ErrHostUndefException,
                    ErrInvalidPortException,
                    ErrConnRefusedException
```

Sends a message to an MHP.

Before executing the `cltSend` method, you must have specified either `DCCLT_ONEWAY_SND` or `DCCLT_SNDRCV` in the `dcsndrcvtype` operand of the TP1/Client/J environment definition, and executed the `rpcOpen` method.

Parameters

`buff`

Specifies the area in which to store the message to be sent. Make sure to specify an area that is at least the size specified in `sendleng`.

`sendleng`

Specifies the length of the message to be sent.

`hostname`

Specifies the host name of the node on which the MHP to be connected exists, in case the connection cannot be established.

If you specify `null`, `cltSend` references the `dcsndhost` operand of the TP1/Client/J environment definition that is obtained when the `rpcOpen` method is executed.

You can also specify an IP address, following the decimal-plus-dot convention, as the host name.

This parameter is ignored if the connection is established.

`portnum`

Specifies an integer for the port number of the MHP to be connected of between 0 and 65535, if the connection is not established.

If you specify `0`, `cltSend` references the `dcsndport` operand of the TP1/Client/J environment definition that is obtained when the `rpcOpen` method is executed.

This parameter is ignored if the connection is established.

`flags`

Specifies whether or not to release the connection after the message is sent.

`DCNOFLAGS`: Does not release the connection after the message is sent.

If `DCNOFLAGS` is specified, unless there is an error, the connection is not released until the `rpcClose` method is executed.

`DCCLT_SND_CLOSE`: Releases the connection after the message is sent.

Return value

None

Exceptions

`ErrInvalidArgsException`

The parameter specification is invalid.

`ErrProtoException`

The `rpcOpen` method did not execute. Another possibility is that, although the `rpcOpen` method did execute, neither `DCCLT_ONEWAY_SND` nor `DCCLT_SNDRCV` is specified in the `dcsndrcvtype` operand of the TP1/Client/J environment definition.

`ErrNetDownException`

A network error occurred.

`ErrSyserrException`

A system error occurred.

`ErrHostUndefException`

The `hostname` parameter specification is invalid. Another possibility is that the host name is not specified in either the `hostname` parameter or in the `dcsndhost` operand in the TP1/Client/J environment definition.

`ErrInvalidPortException`

The `portnum` parameter specification is invalid.

`ErrConnRefusedException`

The connection establishment request to the MHP was denied (an attempt was made to establish a connection through a port that is not waiting for a connection).

Notes

If the MHP drops the connection after the `cltSend` method is executed and a message is being sent, the `cltSend` method next executed ends normally or abnormally. If the `cltSend` method next executed ends normally, the `cltSend` method following that one ends abnormally. We recommend that you take the above into consideration when creating a CUP.

- `cltAssemSend`

```
public void cltAssemSend(byte[] buff,
                         int sendleng,
                         String hostname,
                         int portnum,
                         int timeout,
                         int flags)
                  throws ErrInvalidArgsException,
                         ErrProtoException,
                         ErrNetDownException,
                         ErrSyserrException,
                         ErrHostUndefException,
                         ErrInvalidPortException,
                         ErrConnRefusedException,
                         ErrTimedOutException,
                         ErrConnfreeException,
                         ErrInvalidMessageException,
                         ErrCollisionMessageException
```

Uses the receive message assembly facility to send a message.

To execute the `cltAssemSend` method, you must first specify `DCCLT_ONEWAY_SND` or `DCCLT_SNDRCV` in the `dcsndrcvtype` operand of the TP1/Client/J environment definition and execute the `rpcOpen` method.

When the receive message assembly facility is used, message length (4 + specification value of the `sendleng` argument) is added to the first 4 bytes of the message, and a message having a length of between `buff[0]` and `buff[sendleng - 1]` is sent.

If the connection with the remote system is not established, connection is established according to the values specified in the `hostname` and `portnum` arguments, and a message is sent.

Parameters

`buff`

Specifies the area where the message to be sent is stored. Specify an area that is longer than the value specified by the `sendleng` argument.

`sendleng`

Specifies the length of the message to be sent.

`hostname`

Specifies the host name of the remote system to be connected if no connection has been established.

If `null` is specified, the content of the `dcsndhost` operand of the TP1/Client/J environment definition acquired by executing the `rpcOpen` method is referenced.

For the host name, you can also specify an IP address expressed in the decimal dot method. If connection has been established, this argument is ignored.

`portnum`

Specifies the port number of the remote system to be connected, expressed as an integer of between 0 and 65535, if no connection has been established.

If `0` is specified, the content of the `dcsndport` operand of the TP1/Client/J environment definition acquired by executing the `rpcOpen` method is referenced. If connection has been established, this argument is ignored.

`timeout`

A reserved argument. Specify `0`.

`flags`

Specifies whether to release the connection after the message is sent.

`DCNOFLAGS`: Does not release the connection after the message is sent.

If `DCNOFLAGS` is specified, the connection is not released until the `rpcClose` method is executed, except when an error occurs.

`DCCLT_SND_CLOSE`: Releases the connection after the message is sent.

Return value

None

Exceptions

`ErrInvalidArgsException`

The specified argument is invalid.

`ErrProtoException`

The `rpcOpen` method has not been executed. Another possibility is that, although the `rpcOpen` method has been executed, `DCCLT_ONEWAY_SND` or `DCCLT_SNDRCV` has not been specified in the `dcsndrcvtype` operand of the TP1/Client/J environment definition.

`ErrNetDownException`

A network error occurred.

`ErrSyserrException`

A system error occurred.

`ErrHostUndefException`

The name of the remote host to be connected is invalid. Another possibility is that no host name is specified in either the `hostname` argument or the `dcsndhost` operand of the TP1/Client/J environment definition.

`ErrInvalidPortException`

The specified `portnum` argument is invalid.

`ErrConnRefusedException`

The request to establish connection to the remote system was denied (an attempt was made to establish a connection through a port that is not waiting for a connection).

`ErrTimedOutException`

A timeout occurred while receiving a response message. The connection is released.

`ErrConnfreeException`

The connection was released by the remote system.

`ErrInvalidMessageException`

An invalid message was received.

`ErrCollisionMessageException`

A send/receive message collision occurred.

Note

If the remote system releases the connection when you are sending a message by executing the `cltAssemSend` method, the `cltAssemSend` method that is executed next is terminated either normally or abnormally. If it is terminated normally, abnormal termination occurs with the `cltAssemSend` method that is executed next. Therefore, take note of this fact when creating a CUP.

■ `cltAssemReceive`

```
public void cltAssemReceive(byte[] buff,
                            int[] recvleng,
                            int timeout,
                            int flags)
                     throws ErrInvalidArgsException,
                                              ErrProtoException,
                                              ErrNetDownException,
                                              ErrTimedOutException,
                                              ErrSyserrException,
                                              ErrInvalidPortException,
                                              ErrConnfreeException,
                                              ErrInvalidMessageException,
                                              ErrBufferOverflowException
```

Uses the receive message assembly facility to receive a message.

To execute the `cltAssemReceive` method, you must first specify `DCCLT_ONEWAY_RCV` or `DCCLT_SNDRCV` in the `dcsndrcvtype` operand of the TP1/Client/J environment definition and execute the `rpcOpen` method.

When the receive message assembly facility is used, the first 4 bytes of the message are not stored in the buffer specified by the `buff` argument.

When this method terminates normally, a message containing user data having the length of `recvleng[0]` is received, and the user data is stored in `buff[0]` through `buff[recvleng[0]-1]`.

Parameters

`buff`

Specifies the area in which to store the received message. Specify an area that is longer than the message to be received.

`recvleng`

After the method is executed, the length of the received message is returned to `recvleng[0]`.

`timeout`

Specifies the maximum amount of time to wait to receive a message, expressed as an integer of between -1 and 65535 (seconds).

If `-1` is specified, the system waits indefinitely until a message is received.

If `0` is specified, the system does not wait to receive a message. If there is no message to be received, `ErrTimedOutException` is returned.

If a value between 1 and 65535 is specified, the system waits to receive a message for the number of seconds specified. If a message cannot be received within the specified period, `ErrTimedOutException` is returned.

`flags`

Specifies whether to release the connection after the message is received.

`DCNOFLAGS`: Does not release the connection after the message is received.

If `DCNOFLAGS` is specified, the connection is not released until the `rpcClose` method is executed, except when an error occurs.

`DCCLT_RCV_CLOSE`: Releases the connection after the message is received.

Return value

None

Exceptions

`ErrInvalidArgsException`

The specified argument is invalid.

`ErrProtoException`

The `rpcOpen` method has not been executed. Another possibility is that, although the `rpcOpen` method has been executed, `DCCLT_ONEWAY_RCV` or `DCCLT_SNDRCV` has not been specified in the `dcsndrcvtype` operand of the TP1/Client/J environment definition.

`ErrNetDownException`

A network error occurred.

`ErrTimedOutException`

A timeout occurred while receiving a message.

`ErrSyserrException`

A system error occurred.

`ErrInvalidPortException`

This exception occurs when `Y` is specified in the `dcsockopenatrcv` operand of the TP1/Client/J environment definition, and the port specified in the `dcrcvport` operand is already in use. If `N` is specified for the `dcsockopenatrcv` operand, the `rpcOpen` method returns `ErrFatalException`.

`ErrConnfreeException`

The connection was released by the remote system.

`ErrInvalidMessageException`

An invalid message was received.

`ErrBufferOverflowException`

A message whose length exceeds the value specified in the `buff` argument was received.

Notes

The `cltAssemReceive` method returns control to the CUP in the following cases:

- Message receiving is completed.
- A message whose length exceeds the value specified in the `buff` argument is received.
- A network error occurred.
- A timeout occurred while receiving a message.
- The connection is released by the remote system.
- A message with an invalid message length is received.
- A message with invalid segment information is received.

If the remote system releases the connection when the `cltAssemReceive` method is issued, the method returns the `ErrConnfreeException` error.

■ `setConnectInformation`

```
public void setConnectInformation(byte[] inf,
                                  short inf_len)
                            throws ErrInvalidArgsException
```

Sets the terminal identification information.

The terminal identification information specified in this method takes effect when you specify the host name of a DCCM3 logical terminal in the `dchost` operand and the port number of the DCCM3 logical terminal in the `dchost` or `dcrapport` operand in the TP1/Client/J environment definition, and when you use one of the following methods to establish a permanent connection with the DCCM3 logical terminal:

- Call the `openConnection` method. If the `openConnection` method has parameters, specify DCCM3 logical terminal host name in the `host` parameter and the port number of the DCCM3 logical terminal in the `port` parameter.
- Specify `Y` in the `dcrapautoconnect` operand in the TP1/Client/J environment definition and then call the `rpcCall` method.

When this method is called, the terminal identification information specified in the `dccltconnectinf` operand in the TP1/Client/J environment definition is ignored until the `rpcOpen` method is called again.

The terminal identification information specified in this method is recognized when a permanent connection is established with the DCCM3 logical terminal. If this method is called more than once, the terminal identification information specified immediately before the permanent connection was established with the DCCM3 logical terminal takes effect.

The terminal identification information specified by this method is ignored for an RPC using the name service or scheduler direct facility.

Parameters

`inf`

Specifies the logical terminal name of the DCCM3 logical terminal as terminal identification information consisting of up to 64 bytes of EBCDIK codes. Note that DCCM3 uses only the first 8 bytes and ignores the rest.

`inf_len`

Specifies the length of the terminal identification information, in the range from 1 to 64 bytes.

Return value

None.

Exception

`ErrInvalidArgsException`

A specified parameter is invalid.

■ `acceptNotification`

```
public void acceptNotification(
                                      byte[] inf,
                                      int[] inf_len,
                                      int port,
                                      int timeout,
                                      byte[] hostname,
                                      byte[] nodeid)
                               throws ErrInvalidArgsException,
                                      ErrProtoException,
                                      ErrIOErrException,
                                      ErrSecurityException,
                                      ErrInvalidPortException,
                                      ErrTimedOutException,
                                      ErrNetDownException,
                                      ErrInvalidMessageException,
                                      ErrAcceptCanceledException,
                                      ErrReplyTooBigException,
                                      ErrVersionException,
                                      ErrSyserrException
```

Waits for a message sent by the server's function (`dc_rpc_cltsend`) for up to the amount of time specified in the `timeout` parameter. When the method receives the message, it returns control to the CUP and returns the notification message, length of the notification message, and the host name and node identifier of the server that sent the message.

Parameters

`inf`

Specifies the area for storing the notification message from the server.

When the method terminates normally, the notification message from the server is stored.

`inf_len`

Specifies the length of the area for storing the notification message from the server (length of the area in the `inf` parameter). The value must be in the range from 0 to `DCRPC_MAX_MESSAGE_SIZE`.

If the `dccltrpcmaxmsgsize` value specified in the TP1/Client/J environment definition is `2` or greater, the permitted maximum value is that `dccltrpcmaxmsgsize` value, not the value of `DCRPC_MAX_MESSAGE_SIZE` (1 megabyte).

When the method terminates normally, the length of the notification message from the server is stored.

`port`

Specifies the port number that is to be used to receive notification messages from the server. Specify a value in the range from 5001 to 65535. If you execute multiple processes or threads concurrently on the same machine, specify different port numbers in the `port` parameters. This port number must be unique from port numbers used by the OS or other programs. If a port number that is already in use is specified, response data may not be received correctly. For details about the port numbers used by the OS, see the OS-related documentation.

`timeout`

Specifies the timeout value, in the range from 0 to 65535 (seconds). If `0` is specified, the method waits indefinitely.

`hostname`

Specifies the area for storing the host name of the notifying server. This size of this area must at least 256 bytes.

When the method terminates normally, the host name of the notifying server is stored in this area. TP1/Client/J acquires the host name from the IP address of the notifying server using the `getHostName` method of the `java.net.InetAddress` class and then converts the acquired host name to a byte array using the platform's default character set. TP1/Client/J then stores the result in `hostname`. If the conversion from IP address to host name fails, TP1/Client/J stores the IP address in `hostname` in decimal dot notation (example: `10.209.15.124`).

If `null` is specified, the method does not store the host name of the notifying server.

`nodeid`

Specifies the area for storing the node identifier of the notifying server. The size of this area must be at least 8 bytes.

When the method terminates normally, the node identifier of the notifying server is stored in this area. The format of a node identifier is as follows:

Node identifier (4 bytes)	NULL character (= 0) (4 bytes)

Return value

None.

Exception

`ErrInvalidArgsException`

A parameter specified in the method is invalid.

`ErrProtoException`

The `rpcOpen` method has not been executed.

`ErrIOErrException`

An I/O exception occurred.

`ErrSecurityException`

A security exception occurred.

`ErrInvalidPortException`

The port number specified in the `port` parameter is in use.

`ErrClientTimedOutException`

A timeout occurred on TP1/Client/J.

`ErrNetDownAtClientException`

A network error occurred between TP1/Server and CUP.

`ErrInvalidMessageException`

An invalid message was received.

`ErrAcceptCanceledException`

The unidirectional message reception wait status was released by the `cancelNotification` method. In this case, values have already been set in the `inf`, `inf_len`, and `hostname` parameters. In the `nodeid` parameter, a value whose leading 8 bytes are initialized to `0` is set.

`ErrReplyTooBigException`

The received message cannot fit in the area provided by the CUP. The excess portion of the message was discarded. In this case, values have already been set in the `hostname` and `nodeid` parameters.

`ErrVersionException`

The version of the notifying server is invalid.

`ErrSyserrException`

A system error occurred.

- `cancelNotification`

```
public void cancelNotification(
                               byte[] inf,
                               int inf_len,
                               String hostname,
                               int port)
                        throws ErrInvalidArgsException,
                               ErrProtoException,
                               ErrIOErrException,
                               ErrInvalidPortException,
                               ErrTimedOutException,
                               ErrNetDownException,
                               ErrHostUndefException,
                               ErrSyserrException
```

Releases the unidirectional server message reception wait status (issuance of the `acceptNotification` or `acceptNotificationChained` method). Once this wait status is released, you can send messages specified in the `inf` parameter to the CUP in the unidirectional message reception wait status.

Parameters

`inf`

Specifies the message to be sent to the CUP.

`inf_len`

Specifies the length of the message that is sent to the CUP (length of message set in the `inf` parameter). The value must be in the range from 0 to `DCRPC_MAX_MESSAGE_SIZE`. If `0` is specified, no message is sent to the CUP.

If the `dccltrpcmaxmsgsize` value specified in the TP1/Client/J environment definition is `2` or greater, the permitted maximum value is that `dccltrpcmaxmsgsize` value, not the value of `DCRPC_MAX_MESSAGE_SIZE` (1 megabyte).

`hostname`

Specifies the name of the host where the CUP in the unidirectional message reception wait status is located. For the host name, you can also specify the IP address in decimal dot notation (example: `10.209.15.124`).

`port`

Specifies the port number, in the range from 5001 to 65535, of the CUP that is in the unidirectional message reception wait status.

Return value

None.

Exception

`ErrInvalidArgsException`

A parameter specified in the method is invalid.

`ErrProtoException`

The `rpcOpen` method has not been executed.

`ErrIOErrException`

An I/O exception occurred.

`ErrInvalidPortException`

The port number specified in the `port` parameter is invalid.

`ErrClientTimedOutException`

A timeout occurred on TP1/Client/J.

`ErrNetDownAtClientException`

A network error occurred between CUPs.

`ErrHostUndefException`

The host name specified in the `hostname` parameter is invalid.

`ErrSyserrException`

A system error occurred.

■ `openNotification`
`public void openNotification(int port)`

```
                                throws  ErrInvalidArgsException,
                                        ErrProtoException,
                                        ErrInvalidPortException,
                                        ErrNetDownException,
                                        ErrSyserrException
```

Creates an environment for using the unidirectional message consecutive reception facility. Issue this method as a pair with the `closeNotification` method. Once this method terminates normally, make sure that you issue the `closeNotification` method.

Parameters

`port`

Specifies the port number, in the range from 5001 to 65535, that is to be used to receive notification messages from the server.

If you execute multiple processes or threads concurrently on the same machine, specify different port numbers in the `port` parameters. This port number must be unique from port numbers used by the OS or other programs. If a port number that is already in use is specified, response data may not be received correctly. For details about the port numbers used by the OS, see the OS-related documentation.

Return value

None.

Exception

`ErrInvalidArgsException`

A parameter specified in the method is invalid.

`ErrProtoException`

The `rpcOpen` method has not been executed or the `openNotification` method has already been executed.

`ErrInvalidPortException`

The port number specified in the `port` parameter is already in use.

`ErrNetDownAtClientException`

A network error occurred between TP1/Server and CUP.

`ErrSyserrException`

A system error occurred.

- `acceptNotificationChained`
  `public void acceptNotificationChained(`

```
                          byte[] inf,
                          int[] inf_len,
                          int timeout,
                          byte[] hostname,
                          byte[] nodeid)
                  throws  ErrInvalidArgsException,
                          ErrProtoException,
                          ErrIOErrException,
                          ErrSecurityException,
                          ErrTimedOutException,
                          ErrNetDownException,
                        ErrInvalidMessageException,
                        ErrAcceptCanceledException,
                          ErrReplyTooBigException,
                          ErrVersionException,
                          ErrSyserrException
```

Waits for a message sent by the server's function (`dc_rpc_cltsend`) until the amount of time specified in the `timeout` parameter is reached. When the method receives the message, it returns control to the CUP and returns the notification message, the length of the notification message, and the host name and node identifier of the server that sent the message. In order to issue this method, you must have already issued the `openNotification` method. You must issue this method after the `openNotification` method has been issued but before the `closeNotification` method is issued.

Parameters

`inf`

Specifies the area for storing the notification message from the server.

When the method terminates normally, the notification message from the server is stored.

`inf_len`

Specifies the length of the area for storing the notification message from the server (length of the area in the `inf` parameter). The value must be in the range from 0 to `DCRPC_MAX_MESSAGE_SIZE`.

If the `dccltrpcmaxmsgsize` value specified in the TP1/Client/J environment definition is `2` or greater, the permitted maximum value is that `dccltrpcmaxmsgsize` value, not the value of `DCRPC_MAX_MESSAGE_SIZE` (1 megabyte).

When the method terminates normally, the length of the notification message from the server is stored.

`timeout`

Specifies the timeout value, in the range from 0 to 65535 (seconds). If `0` is specified, the method waits indefinitely.

`hostname`

Specifies the area for storing the host name of the notifying server. This size of this area must be at least 256 bytes.

When the method terminates normally, the host name of the notifying server is stored in this area.

TP1/Client/J acquires the host name from the IP address of the notifying server using the `getHostName` method of the `java.net.InetAddress` class and then converts the acquired host name to a byte array using the platform's default character set. TP1/Client/J then stores the result in `hostname`. If the conversion from IP address to host name fails, TP1/Client/J stores the IP address in `hostname` in decimal dot notation (example: `10.209.15.124`).

If `null` is specified, the method does not store the host name of the notifying server.

`nodeid`

Specifies the area for storing the node identifier of the notifying server. The size of this area must be at least 8 bytes.

When the method terminates normally, the node identifier of the notifying server is stored in this area. The format of a node identifier is as follows:

Node identifier (4 bytes)	NULL character (= 0) (4 bytes)

Return value

None.

Exception

`ErrInvalidArgsException`

A parameter specified in the method is invalid.

`ErrProtoException`

The `openNotification` method has not been executed.

`ErrIOErrException`

An I/O exception occurred.

`ErrSecurityException`

A security exception occurred.

`ErrClientTimedOutException`

A timeout occurred on TP1/Client/J.

`ErrNetDownAtClientException`

A network error occurred between TP1/Server and CUP.

`ErrInvalidMessageException`

An invalid message was received.

`ErrAcceptCanceledException`

The unidirectional message reception wait status was released by the `cancelNotification` method. In this case, values have already been set in the `inf`, `inf_len`, and `hostname` parameters. In the `nodeid` parameter, a value whose leading 8 bytes are initialized to `0` is set.

`ErrReplyTooBigException`

The received message cannot fit in the area provided by the CUP. The excess portion of the message was discarded. In this case, values have already been set in the `hostname` and `nodeid` parameters.

`ErrVersionException`

The version of the notifying server is invalid.

`ErrSyserrException`

A system error occurred.

■ `closeNotification`

```
public void closeNotification()
                      throws  ErrNetDownException,
                              ErrSyserrException
```

Deletes the environment for using the unidirectional message consecutive reception facility. Issue this method as a pair with the `openNotification` method. Once the `openNotification` method has terminated normally, make sure that you issue this method.

Parameters

None.

Return value

None.

Exception

`ErrNetDownAtClientException`

A network error occurred between TP1/Server and CUP.

`ErrSyserrException`

A system error occurred.

Class DCRpcBindTbl

```
java.lang.Object
 └──JP.co.Hitachi.soft.OpenTP1.DCRpcBindTbl
```

```
public class DCRpcBindTbl
extends java.lang.Object
```

This class manages communication-target information. It is used with the `rpcCallTo` method.

Constructor

- `DCRpcBindTbl`
  `public DCRpcBindTbl()`

  Creates an instance of the class that manages communication-target information.

Class ErrAcceptCanceledException

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--JP.co.Hitachi.soft.OpenTP1.TP1ClientException
                    |
                    +--JP.co.Hitachi.soft.OpenTP1.ErrAcceptCanceledException
```

```
public class ErrAcceptCanceledException
extends TP1ClientException
```

This class is an exception returned by TP1/Client/J. It represents the following event:

The unidirectional message reception wait status was released.

Constructor

- `ErrAcceptCanceledException`
  `public ErrAcceptCanceledException()`

  Creates an instance of the `ErrAcceptCanceledException` class without a detail message.

Class ErrBufferOverflowException

```
java.lang.Object
  |
  └── java.lang.Throwable
        |
        └── java.lang.Exception
              |
              └── JP.co.Hitachi.soft.OpenTP1.TP1ClientException
                    |
                    └── JP.co.Hitachi.soft.OpenTP1.ErrBufferOverflowException
```

```
public class ErrBufferOverflowException
extends TP1ClientException
```

This is an exception class returned by TP1/Client/J. It indicates the following event:

The receive buffer overflowed because a message whose length exceeds the receive buffer size specified in the `cltAssemReceive` method was received.

Constructor

■ `ErrBufferOverflowException`
`public ErrBufferOverflowException()`

Creates an instance of the `ErrBufferOverflowException` class without a detail message.

Class ErrCollisionMessageException

```
java.lang.Object
  |
  └── java.lang.Throwable
        |
        └── java.lang.Exception
              |
              └── JP.co.Hitachi.soft.OpenTP1.TP1ClientException
                    |
                    └── JP.co.Hitachi.soft.OpenTP1.ErrCollisionMessageException
```

```
public class ErrCollisionMessageException
extends TP1ClientException
```

This is an exception class returned by TP1/Client/J. It indicates the following event:

A send/receive message collision occurred.

Constructor

- ErrCollisionMessageException
  `public ErrCollisionMessageException()`

  Creates an instance of `ErrCollisionMessageException` without a detail message.

Class ErrClientTimedOutException

```
java.lang.Object
 └─java.lang.Throwable
    └─ java.lang.Exception
        └─ JP.co.Hitachi.soft.OpenTP1.TP1ClientException
            └─ JP.co.Hitachi.soft.OpenTP1.ErrTimedOutException
                └─JP.co.Hitachi.soft.OpenTP1.ErrClientTimedOutException
```

```
public class ErrClientTimedOutException
extends ErrTimedOutException
```

This is an exception class returned by TP1/Client/J. It indicates the following event:

A timeout occurred in the TP1/Client/J system.

Related items

`TP1Client`, `TP1ClientException`, `ErrTimedOutException`

Constructor

- ErrClientTimedOutException
  ```
  public ErrClientTimedOutException()
  ```

  Creates an instance of `ErrClientTimedOutException` without a detail message.

Class ErrConnfreeException

```
java.lang.Object
  └──java.lang.Throwable
        └──java.lang.Exception
              └──JP.co.Hitachi.soft.OpenTP1.TP1ClientException
                    └── JP.co.Hitachi.soft.OpenTP1.ErrConnfreeException
```

```
public class ErrConnfreeException
extends TP1ClientException
```

This is an exception class returned by TP1/Client/J. It indicates the following event:

If an `rpcCall` method returned this exception:

A permanent connection with the RAP-processing server was disconnected. Another possibility is that the virtual session with TP1/Web ended.

If a `trnBegin` method, `trnChainedCommit` method, `trnChainedRollback` method, `trnUnchainedCommit` method, or `trnUnchainedRollback` method returned this exception:

A permanent connection was dropped by the CUP execution process side.

If a `cltReceive` method returned this exception:

The connection was released from the MHP.

If a `cltAssemSend` method or `cltAssemReceive` method returned this exception:

The connection was released by the remote system.

Related items

`TP1Client`, `TP1ClientException`

Constructor

- `ErrConnfreeException`
  `public ErrConnfreeException()`

Creates an instance of `ErrConnfreeException` without a detail message.

Class ErrConnRefusedException

```
java.lang.Object
  └─java.lang.Throwable
      └─java.lang.Exception
          └─JP.co.Hitachi.soft.OpenTP1.TP1ClientException
              └─ JP.co.Hitachi.soft.OpenTP1.ErrConnRefusedException
```

```
public class ErrConnRefusedException
extends Exception
```

This is an exception class returned by TP1/Client/J.

This exception indicates that an error occurred while an attempt was being made to establish a socket connection to a remote address and port.

Generally, this is caused by a connection being denied by the remote side (there is no standby process on the remote address and port).

Related items

`TP1Client`, `TP1ClientException`

Constructor

- `ErrConnRefusedException`
  `public ErrConnRefusedException()`

  Creates an instance of `ErrConnRefusedException` without a detail message.

Class ErrFatalException

```
java.lang.Object
  └─java.lang.Throwable
       └─java.lang.Exception
            └─JP.co.Hitachi.soft.OpenTP1.TP1ClientException
                  └─ JP.co.Hitachi.soft.OpenTP1.ErrFatalException
```

```
public class ErrFatalException
extends TP1ClientException
```

This is an exception class returned by TP1/Client/J. It indicates the following event:

Initialization of a channel failed; or there is a specification error in the TP1/Client/J environment definition.

Related items

`TP1Client`, `TP1ClientException`

Constructor

- `ErrFatalException`
  `public ErrFatalException()`

  Creates an instance of `ErrFatalException` without a detail message.

Class ErrHazardException

```
java.lang.Object
  └─java.lang.Throwable
       └─java.lang.Exception
            └─JP.co.Hitachi.soft.OpenTP1.TP1ClientException
                 └─JP.co.Hitachi.soft.OpenTP1.ErrHazardException
```

```
public class ErrHazardException
extends TP1ClientException
```

This is an exception class returned by TP1/Client/J. It indicates the following event:

A transaction branch of the global transaction was completed heuristically. However, the result of the synchronization point of the transaction branch that was completed heuristically was unknown due to an error. For the result of the synchronization point of the global transaction, resource manager, and UAP that caused this exception, see TP1/Server's message log file. Even after this exception has been returned, this process is still under the transaction and within the range of the global transaction.

Related items

`TP1Client`, `TP1ClientException`

Constructor

- `ErrHazardException`
  `public ErrHazardException()`

Creates an instance of `ErrHazardException` without a detail message.

Class ErrHazardNoBeginException

```
java.lang.Object
  └─java.lang.Throwable
        └─java.lang.Exception
              └─JP.co.Hitachi.soft.OpenTP1.TP1ClientException
                    └─ JP.co.Hitachi.soft.OpenTP1.ErrHazardNoBeginException
```

```
public class ErrHazardNoBeginException
extends TP1ClientException
```

This is an exception class returned by TP1/Client/J. It indicates the following event:

A transaction branch of the global transaction was completed heuristically. However, the result of the synchronization point of the transaction branch that was completed heuristically was unknown due to an error. For the result of the synchronization point of the global transaction, resource manager, and UAP that caused this exception, see TP1/Server's message log file. The method was unable to start a new transaction. Once this exception has been returned, this process is no longer under the transaction.

Related items

`TP1Client`, `TP1ClientException`

Constructor

- `ErrHazardNoBeginException`
  `public ErrHazardNoBeginException()`

  Creates an instance of `ErrHazardNoBeginException` without a detail message.

Class ErrHeuristicException

```
java.lang.Object
  └── java.lang.Throwable
        └── java.lang.Exception
              └── JP.co.Hitachi.soft.OpenTP1.TP1ClientException
                    └── JP.co.Hitachi.soft.OpenTP1.ErrHeuristicException
```

```
public class ErrHeuristicException
extends TP1ClientException
```

This is an exception class returned by TP1/Client/J. It indicates the following event:

For heuristic determination purposes, some of the transaction branches in the global transaction were committed and some were rolled back. This exception is returned when the result of the heuristic determination does not match the result of the synchronization point of the global transaction. For the result of the synchronization point of the global transaction, resource manager, and UAP that caused this exception, see TP1/Server's message log file. Even after this exception has been returned, this process is still under the transaction and within the range of the global transaction.

Related items

`TP1Client`, `TP1ClientException`

Constructor

- `ErrHeuristicException`
  `public ErrHeuristicException()`

  Creates an instance of `ErrHeuristicException` without a detail message.

Class ErrHeuristicNoBeginException

```
java.lang.Object
  └─java.lang.Throwable
      └─java.lang.Exception
          └─JP.co.Hitachi.soft.OpenTP1.TP1ClientException
              └─ JP.co.Hitachi.soft.OpenTP1.ErrHeuristicNoBeginException
```

```
public class ErrHeuristicNoBeginException
extends TP1ClientException
```

This is an exception class returned by TP1/Client/J. It indicates the following event:

For heuristic determination purposes, some of the transaction branches in the global transaction were committed and some were rolled back. This exception is returned when the result of the heuristic determination does not match the result of the synchronization point of the global transaction. For the result of the synchronization point of the global transaction, resource manager, and UAP that caused this exception, see TP1/Server's message log file. For the result of the synchronization point of the global transaction, resource manager, and UAP that caused this exception, see TP1/ Server's message log file. The method was unable to start a new transaction. Once this exception has been returned, this process is no longer under the transaction.

Related items

`TP1Client`, `TP1ClientException`

Constructor

■ `ErrHeuristicNoBeginException`
`public ErrHeuristicNoBeginException()`

Creates an instance of `ErrHeuristicNoBeginException` without a detail message.

Class ErrHostUndefException

```
java.lang.Object
  └─java.lang.Throwable
      └─java.lang.Exception
          └─JP.co.Hitachi.soft.OpenTP1.TP1ClientException
              └─JP.co.Hitachi.soft.OpenTP1.ErrHostUndefException
```

```
public class ErrHostUndefException
extends TP1ClientException
```

This is an exception class returned by TP1/Client/J. It indicates the following event:

If a `setDchost` method returned this exception:

The `host` parameter specification is invalid.

If an `openConnection` method returned this exception:

- The RAP-processing listener host name is not specified in the `dchost` operand of the TP1/Client/J environment definition.
- The URL (information such as the protocol, Web server, CGI name of the prompter, or TP1/Web service name) specified in the `url` parameter or in the `dcweburl` operand of the TP1/Client/J environment definition is invalid.
- The `host` parameter specification is invalid.

If an `rpcCall` method returned this exception:

- Either the host name of the communication target TP1/Server is not specified in the `dchost` operand of the TP1/Client/J environment definition, or the specification is invalid.
- The URL (information such as the protocol, Web server, CGI name of the prompter, or TP1/Web service name) specified in the `dcweburl` operand of the TP1/Client/J environment definition is invalid.

If a `trnBegin` method returned this exception:

Either the host name of the communication target TP1/Server is not specified in the `dchost` operand of the TP1/Client/J environment definition, or the specification is invalid.

If a `cltSend` method or `cltAssemSend` method returned this exception:

The `hostname` parameter specification is invalid. Another possibility is that the host name is not specified in either the `hostname` parameter or in the `dcsndhost` operand in the TP1/Client/J environment definition.

Related items

`TP1Client`, `TP1ClientException`

Constructor

- `ErrHostUndefException`
  `public ErrHostUndefException()`

  Creates an instance of `ErrHostUndefException` without a detail message.

Class ErrInitializingException

```
java.lang.Object
  |
  +--java.lang.Throwable
       |
       +--java.lang.Exception
            |
            +--JP.co.Hitachi.soft.OpenTP1.TP1ClientException
                 |
                 +--JP.co.Hitachi.soft.OpenTP1.ErrInitializingException
```

```
public class ErrInitializingException
extends TP1ClientException
```

This is an exception class returned by TP1/Client/J. It indicates the following event:

The TP1/Server at the node specified in the service request is under start processing.

Related items

`TP1Client`, `TP1ClientException`

Constructor

- `ErrInitializingException`
  `public ErrInitializingException()`

  Creates an instance of `ErrInitializingException` without a detail message.

Class ErrInvalidArgsException

```
java.lang.Object
  └──java.lang.Throwable
       └──java.lang.Exception
            └──JP.co.Hitachi.soft.OpenTP1.TP1ClientException
                 └── JP.co.Hitachi.soft.OpenTP1.ErrInvalidArgsException
```

```
public class ErrInvalidArgsException
extends TP1ClientException
```

This is an exception class returned by TP1/Client/J. It indicates the following event:

An argument specified in the method is invalid.

Related items

`TP1Client`, `TP1ClientException`

Constructors

- `ErrInvalidArgsException`
  `public ErrInvalidArgsException()`

  Creates an instance of `ErrInvalidArgsException` without a detail message.

- `ErrInvalidArgsException`
  `public ErrInvalidArgsException(String msg)`

  Creates an instance of `TP1ClientException` with a detail message. You can use the `getMessage` method to retrieve the detail message.

Class ErrInvalidMessageException

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--JP.co.Hitachi.soft.OpenTP1.TP1ClientException
                    |
                    +--JP.co.Hitachi.soft.OpenTP1.ErrInvalidMessageException
```

```
public class ErrInvalidMessageException
extends TP1ClientException
```

This is an exception class returned by TP1/Client/J. It indicates the following event:

If a `cltAssemReceive` method returned this exception:

> The message length of the received message was not found between `0x00000005` and `0x7FFFFFFF`.
>
> The segment information is invalid.

If a `cltAssemSend` method returned this exception:

> The message length of the message received during receiving of a response is between `0x0000000B` and `0x7FFFFFFF`, or the segment information is invalid.

Constructor

- `ErrInvalidMessageException`
  `public ErrInvalidMessageException()`

  Creates an instance of the `ErrInvalidMessageException` class without a detail message.

Class ErrInvalidPortException

```
java.lang.Object
  └─java.lang.Throwable
      └─java.lang.Exception
          └─JP.co.Hitachi.soft.OpenTP1.TP1ClientException
              └─JP.co.Hitachi.soft.OpenTP1.ErrInvalidPortException
```

```
public class ErrInvalidPortException
extends TP1ClientException
```

This is an exception class returned by TP1/Client/J. It indicates the following event:

If a `setDchost` method, `openConnection` method, or `trnBegin` method returned this exception:

The `port` parameter specification is invalid.

If an `rpcCall` method returned this exception:

- If an RPC that uses a remote API function was executed, the `dcrapport` operand of the TP1/Client/J environment definition is not specified.
- If an RPC that uses a scheduler direct function was executed, the `dcscdport` operand of the TP1/Client/J environment definition is not specified.

If a `cltSend` method or `cltAssemSend` method returned this exception:

The `portnum` parameter specification is invalid.

If a `cltReceive` method or `cltAssemReceive` method returned this exception:

This exception occurs when `Y` is specified in the `dcsockopenatrcv` operand of the TP1/Client/J environment definition, and the port specified in the `dcrcvport` operand is already in use. If `N` is specified for the `dcsockopenatrcv` operand, the `rpcOpen` method returns `ErrFatalException`.

Related items

`TP1Client`, `TP1ClientException`

Constructor

- `ErrInvalidPortException`
  `public ErrInvalidPortException()`

  Creates an instance of `ErrInvalidPortException` without a detail message.

Class ErrInvalidReplyException

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--JP.co.Hitachi.soft.OpenTP1.TP1ClientException
                    |
                    +--JP.co.Hitachi.soft.OpenTP1.ErrInvalidReplyException
```

```
public class ErrInvalidReplyException
extends TP1ClientException
```

This is an exception class returned by TP1/Client/J. It indicates the following event:

The length of the response returned from the service function is not within the range from 1 to the value specified in `DCRPC_MAX_MESSAGE_SIZE`#.

#

When the `dccltrpcmaxmsgsize` operand is used, the value specified in the `dccltrpcmaxmsgsize` operand is used rather than the value specified in `DCRPC_MAX_MESSAGE_SIZE`.

Related items

`TP1Client`, `TP1ClientException`

Constructor

- `ErrInvalidReplyException`
  `public ErrInvalidReplyException()`

Creates an instance of `ErrInvalidReplyException` without a detail message.

Class ErrIOErrException

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--JP.co.Hitachi.soft.OpenTP1.TP1ClientException
                    |
                    +--JP.co.Hitachi.soft.OpenTP1.ErrIOErrException
```

```
public class ErrIOErrException
extends TP1ClientException
```

This is an exception class returned by TP1/Client/J. It indicates the following event:

An I/O exception occurred. For details, see the exceptions and notes for each method.

Related items

`TP1Client`, `TP1ClientException`

Constructor

- `ErrIOErrException`
  `public ErrIOErrException()`

Creates an instance of `ErrIOErrException` without a detail message.

Class ErrMessageTooBigException

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--JP.co.Hitachi.soft.OpenTP1.TP1ClientException
                    |
                    +--JP.co.Hitachi.soft.OpenTP1.ErrMessageTooBigException
```

```
public class ErrMessageTooBigException
extends TP1ClientException
```

This is an exception class returned by TP1/Client/J. It indicates the following event:

The input parameters length specified in the `in_len` argument of the `rpcCall` method exceeds the maximum value.

Related items

`TP1Client`, `TP1ClientException`

Constructor

- `ErrMessageTooBigException`
  `public ErrMessageTooBigException()`

  Creates an instance of `ErrMessageTooBigException` without a detail message.

Class ErrNetDownAtClientException

```
java.lang.Object
  └─ java.lang.Throwable
       └─java.lang.Exception
            └─ JP.co.Hitachi.soft.OpenTP1.TP1ClientException
                 └─JP.co.Hitachi.soft.OpenTP1.ErrNetDownException
                      └─ JP.co.Hitachi.soft.OpenTP1.ErrNetDownAtClientException
```

```
public class ErrNetDownAtClientException
extends ErrNetDownException
```

This is an exception class returned by TP1/Client/J. It indicates the following event:

A network error occurred during communication between TP1/Server and the CUP.

Related items

`TP1Client`, `TP1ClientException`, `ErrNetDownException`

Constructor

■ `ErrNetDownAtClientException`
`public ErrNetDownAtClientException()`

Creates an instance of `ErrNetDownAtClientException` without a detail message.

Class ErrNetDownAtServerException

```
java.lang.Object
  └─ java.lang.Throwable
       └─ java.lang.Exception
            └─ JP.co.Hitachi.soft.OpenTP1.TP1ClientException
                 └─ JP.co.Hitachi.soft.OpenTP1.ErrNetDownException
                      └─ JP.co.Hitachi.soft.OpenTP1.ErrNetDownAtServerException
```

```
public class ErrNetDownAtServerException
extends ErrNetDownException
```

This is an exception class returned by TP1/Client/J. It indicates the following event:

A network error occurred during communication between TP1/Server and the SPP.

Related items

`TP1Client`, `TP1ClientException`, `ErrNetDownException`

Constructor

- `ErrNetDownAtServerException`
  `public ErrNetDownAtServerException()`

  Creates an instance of `ErrNetDownAtServerException` without a detail message.

Class ErrNetDownException

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--JP.co.Hitachi.soft.OpenTP1.TP1ClientException
                    |
                    +--JP.co.Hitachi.soft.OpenTP1.ErrNetDownException
```

Known direct descendant subclasses:
    `ErrNetDownAtClientException`
    `ErrNetDownAtServerException`

```
public class ErrNetDownException
extends TP1ClientException
```

This is an exception class returned by TP1/Client/J. It indicates the following event:

A network error occurred. Another possibility is that the communication-target TP1/Server is not running.

Related items

`TP1Client`, `TP1ClientException`

Constructor

■ `ErrNetDownException`
`public ErrNetDownException()`

Creates an instance of `ErrNetDownException` without a detail message.

Class ErrNoBeginException

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--JP.co.Hitachi.soft.OpenTP1.TP1ClientException
                    |
                    +--JP.co.Hitachi.soft.OpenTP1.ErrNoBeginException
```

```
public class ErrNoBeginException
extends TP1ClientException
```

This is an exception class returned by TP1/Client/J. It indicates the following event:

Commit or rollback processing terminated normally, but the method was unable to start a new transaction. Once this exception has been returned, this process is no longer under the transaction.

Related items

`TP1Client`, `TP1ClientException`

Constructor

■ `ErrNoBeginException`
`public ErrNoBeginException()`

Creates an instance of `ErrNoBeginException` without a detail message.

Class ErrNoBufsAtServerException

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--JP.co.Hitachi.soft.OpenTP1.TP1ClientException
                    |
                    +--JP.co.Hitachi.soft.OpenTP1.ErrNoBufsAtServerException
```

```
public class ErrNoBufsAtServerException
extends TP1ClientException
```

This is an exception class returned by TP1/Client/J. It indicates the following event:

A memory shortage occurred in the specified service.

Related items

`TP1Client`, `TP1ClientException`

Constructor

- `ErrNoBufsAtServerException`
  `public ErrNoBufsAtServerException()`

  Creates an instance of `ErrNoBufsAtServerException` without a detail message.

Class ErrNoBufsException

```
java.lang.Object
  └─java.lang.Throwable
      └─java.lang.Exception
          └─JP.co.Hitachi.soft.OpenTP1.TP1ClientException
              └─ JP.co.Hitachi.soft.OpenTP1.ErrNoBufsException
```

```
public class ErrNoBufsException
extends TP1ClientException
```

This is an exception class returned by TP1/Client/J. It indicates the following event:

A memory shortage occurred.

Related items

`TP1Client`, `TP1ClientException`

Constructor

- `ErrNoBufsException`
  `public ErrNoBufsException()`

  Creates an instance of `ErrNoBufsException` without a detail message.

Class ErrNoSuchServiceException

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--JP.co.Hitachi.soft.OpenTP1.TP1ClientException
                    |
                    +--JP.co.Hitachi.soft.OpenTP1.ErrNoSuchServiceException
```

```
public class ErrNoSuchServiceException
extends TP1ClientException
```

This is an exception class returned by TP1/Client/J. It indicates the following event:

The service name specified in the `service` parameter is undefined.

Related items

`TP1Client`, `TP1ClientException`

Constructor

-
  ```
  ErrNoSuchServiceException
  public ErrNoSuchServiceException()
  ```

  Creates an instance of `ErrNoSuchServiceException` without a detail message.

Class ErrNoSuchServiceGroupException

```
java.lang.Object
  |
  +--java.lang.Throwable
       |
       +--java.lang.Exception
            |
            +--JP.co.Hitachi.soft.OpenTP1.TP1ClientException
                  |
                  +--JP.co.Hitachi.soft.OpenTP1.ErrNoSuchServiceGroupException
```

```
public class ErrNoSuchServiceGroupException
extends TP1ClientException
```

This is an exception class returned by TP1/Client/J. It indicates the following event:

The service group name specified in the `group` parameter is not defined.

Related items

`TP1Client`, `TP1ClientException`

Constructor

- `ErrNoSuchServiceGroupException`
  `public ErrNoSuchServiceGroupException()`

  Creates an instance of `ErrNoSuchServiceGroupException` without a detail message.

Class ErrNotTrnExtendException

```
java.lang.Object
 └─ java.lang.Throwable
     └─ java.lang.Exception
         └─ JP.co.Hitachi.soft.OpenTP1.TP1ClientException
             └─JP.co.Hitachi.soft.OpenTP1.ErrSyserrException
                 └─ JP.co.Hitachi.soft.OpenTP1.ErrNotTrnExtendException
```

```
public class ErrNotTrnExtendException
extends ErrSyserrException
```

This is an exception class returned by TP1/Client/J. It indicates the following event:

A service request with `DCRPC_TPNOTRAN` specified in the `flags` argument was issued after a chained RPC had been used for transaction processing.

Related items

`TP1Client`, `TP1ClientException`, `ErrSyserrException`

Constructor

- `ErrNotTrnExtendException`
  `public ErrNotTrnExtendException()`

  Creates an instance of `ErrNotTrnExtendException` without a detail message.

Class ErrNotUpException

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--JP.co.Hitachi.soft.OpenTP1.TP1ClientException
                    |
                    +--JP.co.Hitachi.soft.OpenTP1.ErrNotUpException
```

```
public class ErrNotUpException
extends TP1ClientException
```

This is an exception class returned by TP1/Client/J. It indicates the following event:

The TP1/Server at the node containing the specified service is not running.

Related items

`TP1Client`, `TP1ClientException`

Constructor

- `ErrNotUpException`
  `public ErrNotUpException()`

  Creates an instance of `ErrNotUpException` without a detail message.

Class ErrProtoException

```
java.lang.Object
  └─java.lang.Throwable
       └─java.lang.Exception
            └─JP.co.Hitachi.soft.OpenTP1.TP1ClientException
                 └─JP.co.Hitachi.soft.OpenTP1.ErrProtoException
```

```
public class ErrProtoException
extends TP1ClientException
```

This is an exception class returned by TP1/Client/J. It indicates the following event:

The method execution order is invalid.

Related items

`TP1Client`, `TP1ClientException`

Constructor

- `ErrProtoException`
  `public ErrProtoException()`

Creates an instance of `ErrProtoException` without a detail message.

Class ErrReplyTooBigException

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--JP.co.Hitachi.soft.OpenTP1.TP1ClientException
                    |
                    +--JP.co.Hitachi.soft.OpenTP1.ErrReplyTooBigException
```

```
public class ErrReplyTooBigException
extends TP1ClientException
```

This is an exception class returned by TP1/Client/J. It indicates the following event:

The length of the response returned from the server exceeds the size of the area provided by the CUP (the value specified in the `out_data` parameter).

Related items

`TP1Client`, `TP1ClientException`

Constructor

- `ErrReplyTooBigException`
  `public ErrReplyTooBigException()`

  Creates an instance of `ErrReplyTooBigException` without a detail message.

Class ErrRMException

```
java.lang.Object
  └─java.lang.Throwable
      └─java.lang.Exception
          └─JP.co.Hitachi.soft.OpenTP1.TP1ClientException
              └─JP.co.Hitachi.soft.OpenTP1.ErrRMException
```

```
public class ErrRMException
extends TP1ClientException
```

This is an exception class returned by TP1/Client/J. It indicates the following event:

An error occurred in the resource manager. The transaction was not started.

Related items

`TP1Client`, `TP1ClientException`

Constructor

- ■ `ErrRMException`
  `public ErrRMException()`

  Creates an instance of `ErrRMException` without a detail message.

Class ErrRollbackException

```
java.lang.Object
  └─java.lang.Throwable
       └─java.lang.Exception
            └─JP.co.Hitachi.soft.OpenTP1.TP1ClientException
                 └─ JP.co.Hitachi.soft.OpenTP1.ErrRollbackException
```

```
public class ErrRollbackException
extends TP1ClientException
```

This is an exception class returned by TP1/Client/J. It indicates the following event:

The current transaction was rolled back without being committed.

When the `trnChainedCommit` method returns this exception, this process is within the range of the transaction. When the `trnUnchainedCommit` method returns this exception, this process is outside the range of the transaction.

Related items

`TP1Client`, `TP1ClientException`

Constructor

- ■ `ErrRollbackException`
  `public ErrRollbackException()`

Creates an instance of `ErrRollbackException` without a detail message.

Class ErrRollbackNoBeginException

```
java.lang.Object
  └── java.lang.Throwable
        └── java.lang.Exception
              └── JP.co.Hitachi.soft.OpenTP1.TP1ClientException
                    └── JP.co.Hitachi.soft.OpenTP1.ErrRollbackNoBeginException
```

```
public class ErrRollbackNoBeginException
extends TP1ClientException
```

This is an exception class returned by TP1/Client/J. It indicates the following event:

A transaction that was to be committed was rolled back without being committed. The method was unable to start a new transaction. Once this exception has been returned, this process is no longer under the transaction.

Related items

`TP1Client`, `TP1ClientException`

Constructor

- `ErrRollbackNoBeginException`
  `public ErrRollbackNoBeginException()`

Creates an instance of `ErrRollbackNoBeginException` without a detail message.

Class ErrSecchkException

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--JP.co.Hitachi.soft.OpenTP1.TP1ClientException
                    |
                    +--JP.co.Hitachi.soft.OpenTP1.ErrSecchkException
```

```
public class ErrSecchkException
extends TP1ClientException
```

This is an exception class returned by TP1/Client/J. It indicates the following event:

The target SPP whose service was requested is protected by the OpenTP1 security facility. The CUP that issued the service request does not have access authority for the SPP.

Related items

`TP1Client`, `TP1ClientException`

Constructor

- `ErrSecchkException`
  `public ErrSecchkException()`

  Creates an instance of `ErrSecchkException` without a detail message.

Class ErrSecurityException

```
java.lang.Object
  └─java.lang.Throwable
      └─java.lang.Exception
          └─JP.co.Hitachi.soft.OpenTP1.TP1ClientException
              └─ JP.co.Hitachi.soft.OpenTP1.ErrSecurityException
```

```
public class ErrSecurityException
extends TP1ClientException
```

This is an exception class returned by TP1/Client/J. It indicates the following event:

A security exception occurred.

Related items

`TP1Client`, `TP1ClientException`

Constructor

- `ErrSecurityException`
  `public ErrSecurityException()`

  Creates an instance of `ErrSecurityException` without a detail message.

Class ErrServerBusyException

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--JP.co.Hitachi.soft.OpenTP1.TP1ClientException
                    |
                    +--JP.co.Hitachi.soft.OpenTP1.ErrServerBusyException
```

```
public class ErrServerBusyException
extends TP1ClientException
```

This is an exception class returned by TP1/Client/J. It indicates the following event:

The target server that receives requests from the socket cannot receive service requests.

Related items

`TP1Client`, `TP1ClientException`

Constructor

- `ErrServerBusyException`
  `public ErrServerBusyException()`

  Creates an instance of `ErrServerBusyException` without a detail message.

Class ErrServerTimedOutException

```
java.lang.Object
  └─ java.lang.Throwable
       └─java.lang.Exception
            └─JP.co.Hitachi.soft.OpenTP1.TP1ClientException
                 └─JP.co.Hitachi.soft.OpenTP1.ErrTimedOutException
                      └─JP.co.Hitachi.soft.OpenTP1.ErrServerTimedOutException
```

```
public class ErrServerTimedOutException
extends ErrTimedOutException
```

This is an exception class returned by TP1/Client/J. It indicates the following event:

A timeout occurred in the TP1/Server system during service execution.

Related items

`TP1Client`, `TP1ClientException`, `ErrTimedOutException`

Constructor

- `ErrServerTimedOutException`
  `public ErrServerTimedOutException()`

  Creates an instance of `ErrServerTimedOutException` without a detail message.

Class ErrServiceClosedException

```
java.lang.Object
  └─java.lang.Throwable
       └─java.lang.Exception
            └─ JP.co.Hitachi.soft.OpenTP1.TP1ClientException
                 └─ JP.co.Hitachi.soft.OpenTP1.ErrServiceClosedException
```

```
public class ErrServiceClosedException
extends TP1ClientException
```

This is an exception class returned by TP1/Client/J. It indicates the following event:

The service group containing the specified service is shut down.

Related items

`TP1Client`, `TP1ClientException`

Constructor

- `ErrServiceClosedException`
  `public ErrServiceClosedException()`

  Creates an instance of `ErrServiceClosedException` without a detail message.

Class ErrServiceNotUpException

```
java.lang.Object
  └─java.lang.Throwable
      └─java.lang.Exception
          └─JP.co.Hitachi.soft.OpenTP1.TP1ClientException
              └─JP.co.Hitachi.soft.OpenTP1.ErrServiceNotUpException
```

```
public class ErrServiceNotUpException
extends TP1ClientException
```

This is an exception class returned by TP1/Client/J. It indicates the following event:

The SPP whose service was requested has not started, or the SPP whose service was requested terminated abnormally before completing the processing.

Related items

`TP1Client`, `TP1ClientException`

Constructor

- `ErrServiceNotUpException`
  `public ErrServiceNotUpException()`

  Creates an instance of `ErrServiceNotUpException` without a detail message.

Class ErrServiceTerminatedException

```
java.lang.Object
  └─java.lang.Throwable
      └─java.lang.Exception
          └─JP.co.Hitachi.soft.OpenTP1.TP1ClientException
              └─JP.co.Hitachi.soft.OpenTP1.ErrServiceTerminatedException
```

```
public class ErrServiceTerminatedException
extends TP1ClientException
```

This is an exception class returned by TP1/Client/J. It indicates the following event:

The SPP whose service was requested terminated abnormally before completing the processing.

Related items

`TP1Client`, `TP1ClientException`

Constructor

- `ErrServiceTerminatedException`
  `public ErrServiceTerminatedException()`

  Creates an instance of `ErrServiceTerminatedException` without a detail message.

Class ErrServiceTerminatingException

```
java.lang.Object
  └─java.lang.Throwable
      └─java.lang.Exception
          └─ JP.co.Hitachi.soft.OpenTP1.TP1ClientException
              └─ JP.co.Hitachi.soft.OpenTP1.ErrServiceTerminatingException
```

```
public class ErrServiceTerminatingException
extends TP1ClientException
```

This is an exception class returned by TP1/Client/J. It indicates the following event:

The specified service is under termination processing.

Related items

`TP1Client`, `TP1ClientException`

Constructor

- `ErrServiceTerminatingException`
  `public ErrServiceTerminatingException()`

  Creates an instance of `ErrServiceTerminatingException` without a detail message.

Class ErrSyserrAtServerException

```
java.lang.Object
  └─java.lang.Throwable
        └─java.lang.Exception
              └─ JP.co.Hitachi.soft.OpenTP1.TP1ClientException
                    └─JP.co.Hitachi.soft.OpenTP1.ErrSyserrAtServerException
```

```
public class ErrSyserrAtServerException
extends TP1ClientException
```

This is an exception class returned by TP1/Client/J. It indicates the following event:

A system error occurred in the specified service.

Related items

`TP1Client`, `TP1ClientException`

Constructor

■ `ErrSyserrAtServerException`
`public ErrSyserrAtServerException()`

Creates an instance of `ErrSyserrAtServerException` without a detail message.

Class ErrSyserrException

```
java.lang.Object
  └─java.lang.Throwable
      └─java.lang.Exception
          └─JP.co.Hitachi.soft.OpenTP1.TP1ClientException
              └─JP.co.Hitachi.soft.OpenTP1.ErrSyserrException
```

Known direct descendant subclasses:
`ErrNotTrnExtendException`
`ErrTrnchkExtendException`

```
public class ErrSyserrException
extends TP1ClientException
```

This is an exception class returned by TP1/Client/J. It indicates the following event:

A system error occurred. For details, see the error trace or memory trace.

For details about the contents of error and memory traces, see *2.11.4 Error trace and memory trace*.

Related items

`TP1Client`, `TP1ClientException`

Constructor

- `ErrSyserrException`
  `public ErrSyserrException()`

Creates an instance of `ErrSyserrException` without a detail message.

Class ErrTestmodeException

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--JP.co.Hitachi.soft.OpenTP1.TP1ClientException
                    |
                    +--JP.co.Hitachi.soft.OpenTP1.ErrTestmodeException
```

```
public class ErrTestmodeException
extends TP1ClientException
```

This is an exception class returned by TP1/Client/J. It indicates the following event:

A service request was issued to an SPP in the test mode.

Related items

`TP1Client`, `TP1ClientException`

Constructor

- `ErrTestmodeException`
  `public ErrTestmodeException()`

  Creates an instance of `ErrTestmodeException` without a detail message.

Class ErrTimedOutException

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +-- java.lang.Exception
              |
              +-- JP.co.Hitachi.soft.OpenTP1.TP1ClientException
                    |
                    +--JP.co.Hitachi.soft.OpenTP1.ErrTimedOutException
```

Known direct descendant subclasses:
`ErrClientTimedOutException`
`ErrServerTimedOutException`

```
public class ErrTimedOutException
extends TP1ClientException
```

This is an exception class returned by TP1/Client/J. It indicates the following event:

If an `openConnection` method returned this exception:

A timeout occurred while a virtual session with TP1/Web was being started or while a connection was being established with a RAP-processing listener.

If a `closeConnection` method returned this exception:

A timeout occurred while a virtual session with TP1/Web was being disconnected or while a connection with a RAP-processing listener or server was being released.

If an `rpcCall` method returned this exception:

A timeout occurred, or the SPP whose service was requested terminated abnormally before the processing was completed.

If a `cltAssemSend` method returned this exception:

A timeout occurred while receiving a response message.

If a `cltAssemReceive` method returned this exception:

A timeout occurred while receiving a message.

If a `trnChainedCommit` method, `trnChainedRollback` method, `trnUnchainedCommit` method, `trnUnchainedRollback` method, or `cltReceive` method returned this exception:

A timeout occurred.

Related items

`TP1Client`, `TP1ClientException`

Constructor

- `ErrTimedOutException`
  `public ErrTimedOutException()`

  Creates an instance of `ErrTimedOutException` without a detail message.

Class ErrTMException

```
java.lang.Object
  └─java.lang.Throwable
       └─java.lang.Exception
            └─JP.co.Hitachi.soft.OpenTP1.TP1ClientException
                 └─ JP.co.Hitachi.soft.OpenTP1.ErrTMException
```

```
public class ErrTMException
extends TP1ClientException
```

This is an exception class returned by TP1/Client/J. It indicates the following event:

Because the transaction service resulted in an error, the applicable method was unable to start the transaction. When this exception has been returned, you may be able to start the transaction by re-executing the method after waiting a while.

Related items

`TP1Client`, `TP1ClientException`

Constructor

- ■ `ErrTMException`
  `public ErrTMException()`

Creates an instance of `ErrTMException` without a detail message.

Class ErrTrnchkException

```
java.lang.Object
  └─java.lang.Throwable
       └─ java.lang.Exception
            └─ JP.co.Hitachi.soft.OpenTP1.TP1ClientException
                 └─ JP.co.Hitachi.soft.OpenTP1.ErrTrnchkException
```

```
public class ErrTrnchkException
extends TP1ClientException
```

This is an exception class returned by TP1/Client/J. It indicates the following event:

With the inter-node load-balancing facility being used, multiple SPPs did not have the matching transaction attribute. Alternatively, the inter-node load-balancing facility cannot be executed because the TP1/Server version used at the node where the workload is to be balanced is older than the TP1/Client/J version.

This exception is returned only when a service request is issued to an SPP using the inter-node load-balancing facility.

Related items

`TP1Client`, `TP1ClientException`

Constructor

- `ErrTrnchkException`
  `public ErrTrnchkException()`

Creates an instance of `ErrTrnchkException` without a detail message.

Class ErrTrnchkExtendException

```
java.lang.Object
  └─ java.lang.Throwable
       └─java.lang.Exception
            └─ JP.co.Hitachi.soft.OpenTP1.TP1ClientException
                 └─JP.co.Hitachi.soft.OpenTP1.ErrSyserrException
                      └─ JP.co.Hitachi.soft.OpenTP1.ErrTrnchkExtendException
```

```
public class ErrTrnchkExtendException
extends ErrSyserrException
```

This is an exception class returned by TP1/Client/J. It indicates the following event:

Possible causes are as follows:

- No more transaction branches can be started because the number of transaction branches that can be active at the same time has reached the maximum.
- No more transaction branches can be started because the number of child transaction branches that can be started from a single transaction branch has reached the maximum.
- In the case of a service request with domain qualification, `DCRPC_TPNOTRAN` is not specified in the `flags` argument.

Related items

`TP1Client`, `TP1ClientException`, `ErrSyserrException`

Constructor

- `ErrTrnchkExtendException`
  `public ErrTrnchkExtendException()`

Creates an instance of `ErrTrnchkExtendException` without a detail message.

Class ErrVersionException

```
java.lang.Object
  |
  +--java.lang.Throwable
       |
       +--java.lang.Exception
            |
            +--JP.co.Hitachi.soft.OpenTP1.TP1ClientException
                 |
                 +--JP.co.Hitachi.soft.OpenTP1.ErrVersionException
```

```
public class ErrVersionException
extends TP1ClientException
```

This class is an exception returned by TP1/Client/J. It represents the following event:

The version of the notifying server is invalid.

Constructor

- `ErrVersionException`
  `public ErrVersionException()`

  Creates an instance of the `ErrVersionException` class without a detail message.

Class TP1ClientException

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--JP.co.Hitachi.soft.OpenTP1.TP1ClientException
```

Known direct descendant subclasses:

```
ErrBufferOverflowException
ErrCollisionMessageException
ErrConnfreeException
ErrFatalException
ErrHazardException
ErrHazardNoBeginException
ErrHeuristicException
ErrHeuristicNoBeginException
ErrHostUndefException
ErrInitializingException
ErrInvalidArgsException
ErrInvalidMessageException
ErrInvalidPortException
ErrInvalidReplyException
ErrIOErrException
ErrMessageTooBigException
ErrNetDownException
ErrNoBeginException
ErrNoBufsAtServerException
ErrNoBufsException
ErrNoSuchServiceException
ErrNoSuchServiceGroupException
ErrNotUpException
ErrProtoException
ErrReplyTooBigException
ErrRMException
ErrRollbackException
ErrRollbackNoBeginException
ErrSecchkException
ErrSecurityException
ErrServerBusyException
ErrServiceClosedException
ErrServiceNotUpException
```

```
ErrServiceTerminatedException
ErrServiceTerminatingException
ErrSyserrAtServerException
ErrSyserrException
ErrTestmodeException
ErrTimedOutException
ErrTMException
ErrTrnchkException
```

```
public class TP1ClientException
extends java.lang.Exception
```

`TP1ClientException` is the superclass of all exceptions that are returned by TP1/Client/J.

Related item

`TP1Client`

Constructors

- `TP1ClientException`
  `public TP1ClientException()`

  Creates an instance of `TP1ClientException` without a detail message.
- `TP1ClientException`
  `public TP1ClientException(String msg)`

  Creates an instance of `TP1ClientException` with a detail message.

  You can use the `getMessage` method to retrieve the detail message.

Chapter

5. Definitions

This chapter describes the TP1/Client/J environment definitions.

5.1 Overview of definitions
5.2 Details of TP1/Client/J environment definitions

5.1 Overview of definitions

This section provides a list of the TP1/Client/J environment definitions, describes the rules for definitions, and shows the path name format.

5.1.1 List of TP1/Client/J environment definitions

The following table lists and describes the TP1/Client/J environment definitions.

Table 5-1: List of TP1/Client/J environment definitions

No.	Operand	Description	Specification value
1	`dcnamuse`	Specifies whether or not RPCs that use the name service are used	`Y`|<<`N`>>
2	`dcnamport`	Specifies the port number of the name service	<unsigned integer> ((5001-65535)) <<10000>>
3	`dchost`	Specifies the TP1/Server as a gateway	<character string>
4	`dcwatchtim`	Specifies the maximum time to wait for a response	<unsigned integer> ((0-65535)) <<180>> (seconds)
5	`dccltinquiretime`	Specifies the maximum time interval in permanent connection inquiry to another	<unsigned integer> ((0-1048575)) (seconds)
6	`dcwatchtiminherit`	Specifies whether or not the RAP-processing server inherits the CUP's maximum time to wait for a response when an RPC that uses the remote API facility is issued	`Y`|<<`N`>>
7	`dcwatchtimrpcinherit`	Specifies whether or not the TP1/Server inherits the CUP's maximum time to wait for a response	`Y`|<<`N`>>
8	`dccltdelay`	Specifies the maximum communication delay time	<unsigned integer> ((0-65535)) <<0>> (seconds)
9	`dcselint`	Specifies the reply message monitoring interval	<unsigned integer> ((1-65535)) <<100>> (milliseconds)
10	`dccltextend`	Specifies the facility extension level of TP1/Client/J	`00000000`|`00000001`
11	`dccache`	Specifies the maximum number of entries in the name cache	<unsigned integer> ((2-256)) <<8>>

No.	Operand	Description	Specification value
12	`dchostselect`	Specifies whether or not to randomly select a TP1/Server as a gateway	Y|<<N>>
13	`dcscddirect`	Specifies whether or not RPCs that use the scheduler direct facility without querying information to the name service of TP1/Server are used	Y|<<N>>
14	`dcscdport`	Specifies the port number of the schedule service	<unsigned integer> ((5001-65535))
15	`dcscdloadpriority`	Specifies whether or not the TP1/Server that accepts service requests as a gateway has priority for load-balance	Y|<<N>>
16	`dcrapdirect`	Specifies whether or not the facility for directly querying the RAP-processing listener without querying the information to the TP1/Server's name service (remote API facility) is used	<<Y>>|N
17	`dcrapport`	Specifies the port number of the RAP-processing listener	<unsigned integer> ((5001-65535))
18	`dcrapautoconnect`	Specifies whether or not a permanent connection is established automatically between a CUP and a RAP-processing server	Y|<<N>>
19	`dcerrtrace`	Specifies whether or not the CUP collects an error trace	Y|<<N>>
20	`dcerrtracepath`	Specifies the directory for error trace files	<character string>
21	`dcerrtracesize`	Specifies the size of an error trace file	<unsigned integer> ((4096-1048576)) <<4096>> (bytes)
22	`dcmethodtrace`	Specifies whether or not the CUP collects a method trace	Y|<<N>>
23	`dcmethodtracepath`	Specifies the directory for method trace files	<character string>
24	`dcmethodtracesize`	Specifies the size of a method trace file	<unsigned integer> ((4096-1048576)) <<4096>> (bytes)
25	`dcuaptrace`	Specifies whether or not the CUP collects a UAP trace	Y|<<N>>
26	`dcuaptracepath`	Specifies the directory for UAP trace files	<character string>

No.	Operand	Description	Specification value
27	`dcuaptracesize`	Specifies the size of a UAP trace file	<unsigned integer> ((4096-1048576)) <<4096>> (bytes)
28	`dcdatatrace`	Specifies whether or not the CUP collects a data trace	`Y`|<<`N`>>
29	`dcdatatracepath`	Specifies the directory for data trace files	<character string>
30	`dcdatatracesize`	Specifies the size of a data trace file	<unsigned integer> ((4096-1048576)) <<4096>> (bytes)
31	`dcdatatracemaxsize`	Specifies the maximum data size for a data trace	<unsigned integer> ((16-1048576)) <<128>> (bytes)
32	`dccltrstatisitem`	Specifies the item of statistical information to be acquired for transaction branches	*statistical-information-item*`[,`*statistical-information-item*`]...`
33	`dccltroptiitem`	Specifies the optimization items to be used to improve the performance of a global transaction that consists of multiple user servers	*transaction-optimization-item*`[,`*transaction-optimization-item*`]...`
34	`dccltrwatchtime`	Specifies the maximum wait time for communications during transaction synchronization point processing	<unsigned integer> ((1-65535)) (seconds)
35	`dccltrrbinfo`	Specifies whether or not information about the cause of the rollback is logged when a transaction branch rolls back	`no`|`self`|`remote`|`all`
36	`dccltrlimittime`	Specifies the maximum executable time for a transaction branch	<unsigned integer> ((0-65535)) (seconds)
37	`dccltrrbrcv`	Specifies whether or not a rollback completion report is received after sending a rollback instruction to RPC destination transaction branch	`Y`|`N`
38	`dccltrrecoverytype`	Specifies the synchronization point processing method to use in the event of a UAP error	`type1`|`type2`|`type3`
39	`dccltrexptm`	Specifies the expiry time in a transaction branch	<unsigned integer> ((0-65535)) (seconds)
40	`dccltrcputm`	Specifies the transaction branch CPU monitoring time	<unsigned integer> ((0-65535)) (seconds)

No.	Operand	Description	Specification value
41	`dccltrexpsp`	Specifies whether the monitoring time includes the time of subsequent types of processing when transaction branch processing is monitored	`Y`|`N`|`F`
42	`dcsndrcvtype`	Specifies the environment to initialize when the TCP/IP communication facility is used	`DCCLT_ONEWAY_SND`|`DCCLT_ONEWAY_RCV`|`DCCLT_SNDRCV`
43	`dcrcvport`	Specifies the port number of the receiving CUP	<unsigned integer> ((1-65535)) <<11000>>
44	`dcsndhost`	Specifies the host name of the node on which the MHP to be connected exists	<character string>
45	`dcsndport`	Specifies the port number of the MHP to be connected	<unsigned integer> ((1-65535)) <<12000>>
46	`dcsockopenatrcv`	Specifies the trigger for opening a reception socket (the trigger to start waiting to receive a connection from a remote computer) when sending and receiving is performed over a single connection while the TCP/IP communication facility is being used	`Y`|<<`N`>>
47	`dcweburl`	Specifies the URL of the TP1/Web whose service is requested	*<path-name>*
48	`dccltdbgtrcfilecount`	Specifies the maximum number of debug trace files	<unsigned integer> ((0-256)) <<0>>
49	`dccltrpcmaxmsgsize`	Specifies the maximum size of a message sent or received by RPC	<unsigned integer> ((1-8)) <<1>> (megabytes)
50	`dcscdhostchange`	Specifies whether or not to distribute to the RPCs the service request target schedulers specified in the `dchost` operand of the TP1/Client/J environment definition	`Y`|<<`N`>>
51	`dccltloadbalance`	Specifies whether or not to store in cache the information on the service request target scheduler with the smallest workload, from the information on multiple service request target schedulers obtained from the name server	`Y`|<<`N`>>
52	`dccltcachetim`	Specifies the cache expiration time	<unsigned integer> ((0-65535)) <<30>> (seconds)

No.	Operand	Description	Specification value
53	`dccltconnecttimeout`	Specifies the maximum amount of length of time to wait until the connection is established	<unsigned integer> ((0-65535)) <<0>> (seconds)
54	`dccltprftrace`	Specifies whether or not to collect a performance analysis trace of the Cosminexus Application Server when TP1/Client/J is operated on a Cosminexus Application Server	<<`Y`>>|`N`
55	`dccltprfinfosend`	Specifies whether or not to add identification information in TP1/Client/J to the information that is output to the performance verification trace of OpenTP1, when an RPC that uses the name service or scheduler direct facility is issued to TP1/Server	<<`Y`>>|`N`
56	`dccltnammlthost`	Specifies whether or not to issue an RPC that uses the name service to a TP1/ Server in the multi-homed host environment	`Y`|<<`N`>>
57	`dccltdatacomp`	Specifies whether or not the data compression facility is used	`Y`|<<`N`>>
58	`dccltconnectinf`	Specifies (in EBCDIK codes) the logical terminal name of the DCCM3 logical terminal as the terminal identification information	*terminal-identification-information*
59	`dcscdmulti`	Specifies whether or not the multi-scheduler facility is used	`Y`|<<`N`>>
60	`dcscdmulticount`	Specifies the number of processes for the multi-scheduler daemon	<unsigned integer> ((1-4096)) <<1>>
61	`dccltcupsndhost`	Specifies the CUP transmission source host	<character string>
62	`dccltcuprcvport`	Specifies the port number of the CUP that receives messages from the server	<unsigned integer> ((5001-65535))
63	`dcclttrcmplmttm`	Specifies the maximum amount of time to allow for completion of a transaction	<unsigned integer> ((0-65535)) (seconds)

5.1.2 Rules for definitions

This section explains the conventions used to explain the definitions.

The syntax, attribute, and element symbols explained here are not actually specified in the definitions.

(1) Syntax symbols

The syntax symbols are used to explain the syntax.

Symbol	Convention
[ ]	Square brackets enclose an item or set of items whose specification is optional. Example: *host-name*`[:`*port-number*`]` Either *host-name* or *host-name*`:`*port-number* must be specified.
|	Only one of the options separated by a vertical bar can be specified at the same time. Example: `dcnamuse=Y|N` Either `dcnamuse=Y` or `dcnamuse=N` can be specified.
...	An ellipsis (...) indicates that the item or items immediately preceding the ellipsis may be specified as many times as necessary. Example: *host-name*`[:`*port-number*`][,`*host-name*`[:`*port-number*`],...]` *host-name*`[:`*port-number*`]` can be specified as many times as necessary.

(2) Attribute symbols

The attribute symbols are used to explain the attributes of user-specified values, such as a value range.

Attribute symbol	Convention
~	This symbol is followed by the attribute of a user-specified value.
<< >>	Double diamond brackets enclose the default value.
< >	Single diamond brackets enclose the element symbol for a user-specified value.
(( ))	Double parentheses enclose the permitted range of a user-specified value.

(3) Element symbols

The element symbols are used to explain the type of a user-specified value.

Element symbol	Convention
<alphabetic>	Alphabetic characters (A-Z, a-z) and underscore (_)
<alphanumeric>	Alphabetic and numeric characters (0-9)
<alphabetic symbol>	Alphabetic characters (A-Z, a-z), #, @, and \
<unsigned integer>	Numeric characters (0-9)
<unsigned hexadecimal integer>	Numeric characters (0-9), A-F, a-f

Element symbol	Convention
<symbolic name>	String of alphabetic symbols and numeric characters (beginning with an alphabetic symbol)
<character string>	String of any characters
<path name>	Symbolic name, /, and . (period) (The path name depends on the operating system being used.)

5.1.3 Format of path names

When you specify a path name in any of the following operands and you use the `\` as the file delimiter, you must specify two consecutive backslashes (`\\`) to represent a single backslash:

- `dcerrtracepath`
- `dcmethodtracepath`
- `dcuaptracepath`
- `dcdatatracepath`

Example: When collecting error trace in `C:\Clt4J\trace`

```
dcerrtracepath = C:\\Clt4J\\trace
```

or

```
dcerrtracepath = C:/Clt4J/trace
```

5.2 Details of TP1/Client/J environment definitions

For a Java application or servlet, store the TP1/Client/J environment definition in a desired file and specify its file name in the argument of the `rpcOpen` method. If you call the `rpcOpen` method with no argument specified, you can also specify the TP1/Client/J environment definition as system properties.

For a Java applet, specify the TP1/Client/J environment definition using the `param` tags.

The following section presents the TP1/Client/J environment definition.

5.2.1 Format

```
[dcnamuse=Y|N]
[dcnamport=port-number-of-name-service]
[dchost=host-name-of-TP1/Server-as-a-gateway]
[dcwatchtim=maximum-time-to-wait-for-a-response]
[dccltinquiretime=maximum-permanent-connection-inquiry-interval]
[dcwatchtiminherit=Y|N]
[dcwatchtimrpcinherit=Y|N]
[dccltdelay=maximum-communication-delay-time]
[dcselint=reply-text-monitoring-interval]
[dccltextend=facility-extension-level]
[dccache=maximum-entries-count-in-name-cache]
[dchostselect=Y|N]
[dcscddirect=Y|N]
[dcscdport=port-number-of-schedule-service]
[dcscdloadpriority=Y|N]
[dcrapdirect=Y|N]
[dcrapport=port-number-of-RAP-processing-listener]
[dcrapautoconnect=Y|N]
[dcerrtrace=Y|N]
[dcerrtracepath=directory-for-error-trace-file]
[dcerrtracesize=error-trace-file-size]
[dcmethodtrace=Y|N]
[dcmethodtracepath=directory-for-method-trace-file]
[dcmethodtracesize=method-trace-file-size]
[dcuaptrace=Y|N]
[dcuaptracepath=directory-for-UAP-trace-file]
[dcuaptracesize=UAP-trace-file-size]
[dcdatatrace=Y|N]
[dcdatatracepath=directory-for-data-trace-file]
[dcdatatracesize=data-trace-file-size]
[dcdatatracemaxsize=maximum-data-size-for-data-trace]
```

```
[dcclttrstatisitem=statistical-information-item]
[dcclttroptiitem=transaction-optimization-item]
[dcclttrwatchtime=maximum-communication-wait-time-during-transaction-syn
chronization-point-processing]
[dcclttrrbinfo=no|self|remote|all]
[dcclttrlimittime=maximum-executable-time-for-transaction-branch]
[dcclttrrbrcv=Y|N]
[dcclttrrecoverytype=type1|type2|type3]
[dcclttrexptm=expiry-time-in-transaction-branch]
[dcclttrcputm=transaction-branch-CUP-monitoring-time]
[dcclttrexpsp=Y|N|F]
[dcsndrcvtype=DCCLT_ONEWAY_SND|DCCLT_ONEWAY_RCV|DCCLT_SNDRCV]
[dcrcvport=port-number-of-the-receiving-CUP]
[dcsndhost=host-name-of-node-on-which-the-MHP-to-be-connected-exists]
[dcsndport=port-number-of-the-MHP-to-be-connected]
[dcsockopenatrcv=Y|N]
[dcweburl=URL-of-TP1/Web-whose-service-is-requested]
[dccltdbgtrcfilecount=maximum-number-of-debug-trace-files]
[dccltrpcmaxmsgsize=maximum-size-of-messages-sent-or-received-by-RPC]
[dcscdhostchange=Y|N]
[dccltloadbalance=Y|N]
[dccltcachetim=cache-expiration-time]
[dccltconnecttimeout=maximum-time-for-monitoring-connection-establishmen
t]
[dccltprftrace=Y|N]
[dccltprfinfosend=Y|N]
[dccltnammlthost=Y|N]
[dccltdatacomp=Y|N]
[dccltconnectinf=terminal-identification-information]
[dcscdmulti=Y|N]
[dcscdmulticount=processes-count-for-multi-scheduler-daemon]
[dccltcupsndhost=transmission-source-host]
[dccltcuprcvport=port-number-used-for-reception-of-CUP]
[dcclttrcmplmttm=maximum-time-allowed-for-completion-of-transaction]
```

5.2.2 Operands

- `dcnamuse=Y|N` ~<<N>>

  Specifies whether or not RPCs that use the name service are used.

  `Y`: Use RPCs that use the name service.

  `N`: Do not use RPCs that use the name service.

  The `dcnamuse`, `dcrapdirect`, and `dcscddirect` operands are mutually exclusive. You cannot specify `Y` for more than one of these operands at the same time; doing so results in a definition error.

The TP1/Web connection facility can be used only if you specify `Y` in the `dcrapdirect` operand. Defining the `dcweburl` operand and specifying `dcscddirect=Y` and `dcrapdirect=Y` results in a definition error.

- `dcnamport=`*port-number-of-name-service*

  ~<unsigned integer> ((5001~65535)) <<10000>>

  Specifies the port number of the name service that was specified in the `name_port` operand of the system common definition for the OpenTP1 to which the client belongs. This value takes effect on a host for which no port number is specified in the `dchost` operand.

- `dchost=`*host-name-of-TP1/Server-as-a-gateway* ~<character string>

  Specifies the host name of the TP1/Server as a gateway and the port number of the RAP-processing listener, schedule service, or name service. If you use a delimiter (,), you can specify multiple TP1/Servers. To issue RPCs via a RAP-processing listener, specify the host name of the RAP-processing listener or the host name of the firewall.

  If both the `dchost` operand and the `dcweburl` operand are specified, the `dcweburl` operand takes precedence.

  Format

  When `dcrapdirect=Y` is specified:

  host-name`[:`*port-number-of-RAP-processing-listener*`][,`*host-name*`[:`*port-number-of-RAP-processing-listener*`],...]`

  When `dcscddirect=Y` is specified:

  host-name`[:`*port-number-of-schedule-service*`][,`*host-name*`[:`*port-number-of-schedule-service*`],...]`

  When `dcnamuse=Y` is specified:

  host-name`[:`*port-number-of-name-service*`][,`*host-name*`[:`*port-number-of-name-service*`],...]`

  - *host-name* ~<character string>

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

  You can specify a maximum of 256 characters. Do not enter any blank characters (spaces or tabs), except after a delimiter (,). For the host name, you can also specify the IP address expressed in decimal dot format.

  If you specify more than one host name for TP1/Server and a TP1/Server error is detected, TP1/Client/J attempts swapping by referencing the next

TP1/Server specified in the `dchost` operand. However, if RPCs are issued via a RAP-processing listener, TP1/Client/J attempts swapping by referencing the next TP1/Server specified in the `dchost` operand only when `dcrapautoconnect=Y` is also specified in the TP1/Client/J environment definition or `DCRPC_RAP_AUTOCONNECT` is specified in the `setDccltextend` method.

For the port number, if you have specified `dcrapdirect=Y`, specify the port number of the RAP-processing listener. If you have specified `dcscddirect=Y`, specify the port number of the schedule service. If you have specified `dcnamuse=Y`, specify the port number of the name service.

If the port number is omitted and `dcrapdirect=Y` is specified, TP1/Client/J assumes the value of the `dcrapport` operand; if `dcscddirect=Y` is specified, TP1/Client/J assumes the value of the `dcscdport` operand; if `dcnamuse=Y` is specified, TP1/Client/J assumes the value of the `dcnamport` operand.

- `dcwatchtim=`*maximum-time-to-wait-for-a-response*

  ~<unsigned integer> ((0~65535)) <<180>> (seconds)

  Specifies the maximum time to wait for a response from the service since a service request was issued from the CUP to the TP1/Server.

  If no response is returned within the specified time, control is returned to the CUP with an error.

  If 0 is specified, TP1/Client/J waits indefinitely for a response.

  The specification of the `dcwatchtim` operand is applied to the following APIs (methods):

  - `closeConnection` method
  - `openConnection` method
  - `rpcCall` method (except for non-response type RPCs that use the scheduler direct facility)
  - `rpcCallTo` method (for synchronous-response RPCs)
  - `trnBegin` method
  - `trnChainedCommit` method
  - `trnChainedRollback` method
  - `trnUnchainedCommit` method
  - `trnUnchainedRollback` method

- `dccltinquiretime=`*maximum-permanent-connection-inquiry-interval*

~<unsigned integer> ((0~1048575)) (seconds)

Specifies the maximum amount of time from one inquiry to another that is issued from a CUP to the server. The maximum permanent connection inquiry interval is the timer value that is monitored by the RAP-processing server. If there is no inquiry within the specified time, the RAP-processing server releases the permanent connection forcibly. If the maximum permanent connection inquiry interval is reached within a transaction, TP1/Client/J rolls back the corresponding transaction forcibly.

If you specify 0 in this operand, TP1/Client/J waits indefinitely for a response from the CUP. If you omit this operand, the `rap_inquire_time` operand in the RAP-processing listener service definition takes effect.

- `dcwatchtiminherit=Y|N` ~<<N>>

  Specifies whether or not the RAP-processing server is to inherit the CUP's maximum amount of time to wait for a response when an RPC that uses the remote API facility is issued.

  `Y`: The RAP-processing server is to inherit the CUP's maximum time to wait for a response.

  `N`: The RAP-processing server is not to inherit the CUP's maximum time to wait for a response.

- `dcwatchtimrpcinherit=Y|N` ~<<N>>

  Specifies whether or not the TP1/Server is to inherit the CUP's maximum amount of time to wait for a response.

  `Y`: TP1/Server is to inherit the CUP's maximum time to wait for a response.

  `N`: TP1/Server is not to inherit the CUP's maximum time to wait for a response.

- `dccltdelay=`*maximum-communication-delay-time*

  ~<unsigned integer> ((0~65535)) <<0>> (seconds)

  Taking into account the overhead involved with communication between the CUP and the RAP-processing server, you can specify this operand to end response monitoring in the RAP-processing server system earlier than in the TP1/Client/J system. This definition ends response monitoring in the RAP-processing server system early by the specified amount of time in order to avoid missing a message that may arrive after a timeout occurs in the TP1/Client/J system.

  This definition is applicable only when `dcrapdirect=Y` and `dcwatchtiminherit=Y` are specified in the TP1/Client/J environment definition.

  TP1/Client/J ignores the `dccltdelay` operand when `dcwatchtim=0` is specified in the TP1/Client/J environment definition, 0 is specified in the `setDcwatchtim` method, or `dcwatchtiminherit=N` is specified in the TP1/Client/J environment definition. If the `dcwatchtim` operand value minus the `dccltdelay` operand value equals 0 or a negative value, TP1/Client/J also ignores the `dccltdelay` operand, in which case TP1/Client/J assumes a value of 1.

- `dcselint=`*reply-text-monitoring-interval* ~<unsigned integer> ((1~65535)) <<100>> (milliseconds)

  This operand is provided only to provide compatibility with the source of the previous version. Currently, TP1/Client/J ignores this operand, if specified.

- `dccltextend=00000000|00000001`

  Specifies the facility extension level of TP1/Client/J.

  `00000000`: Do not extend the TP1/Client/J facility.

  `00000001`: Send the IP address of the local CUP to the service when the `rpcCall` method is called. You specify this value if you need to obtain the address of the CUP by having the called service execute the `dc_rpc_get_callers_address()` function.

- `dccache=`*maximum-entries-count-in-name-cache* ~<unsigned integer> ((2-256)) <<8>>

  Specifies the maximum number of entries in the cache that is used to cache service information used by TP1/Client/J. This cache is managed by the LRU method. When the number of entries in cache exceeds the specified value, TP1/Client/J deletes the oldest service information that was called in the past.

- `dchostselect=Y|N` ~<<N>>

  Specifies whether or not to randomly select a TP1/Server as a gateway. This definition is applicable only when more than one TP1/Server is specified as a gateway in the `dchost` operand.

  `Y`: Select a TP1/Server as a gateway at random.

  `N`: Do not select a TP1/Server as a gateway at random.

  When `Y` is specified

  > If an error is detected while an inquiry is being made to the name service of the TP1/Server as a gateway, TP1/Client/J randomly selects another TP1/Server for swapping.

  When `N` is specified

  > If an error is detected while an inquiry is being made to the name service of the TP1/Server as a gateway, TP1/Client/J attempts to switch to the next TP1/Server specified in the `dchost` operand.

The erroneous TP1/Server becomes a candidate for a retry-target host for the next RPC.

If `dcscddirect=Y` is specified in the TP1/Client/J environment definition, swapping of gateway TP1/Servers occurs when transmission to the corresponding port number fails.

- `dcscddirect=Y|N` ~<<`N`>>

  Specifies whether or not RPCs that use the scheduler direct facility without querying information to the name service of TP1/Server are used.

  `Y`: Use RPCs that use the scheduler direct facility.

  `N`: Do not use RPCs that use the scheduler direct facility.

  If you have specified the port number of the schedule service in the `dcscdport` operand of the TP1/Client/J environment definition, TP1/Client/J uses that port number for querying. The `dcrapdirect`, `dcscddirect`, and `dcnamuse` operands are mutually exclusive. You cannot specify `Y` for more than one of these operands at the same time; doing so results in a definition error.

  The TP1/Web connection facility can be used only if you specify `Y` in the `dcrapdirect` operand. Defining the `dcweburl` operand and specifying `dcscddirect=Y` and `dcrapdirect=Y` results in a definition error.

- `dcscdport=`*port-number-of-schedule-service*

  ~<unsigned integer> ((5001~65535))

  Specifies the port number of the schedule service that has been specified in the `scd_port` operand of the schedule service definition for the OpenTP1 to which the client belongs. This value takes effect on a host for which no port number has been specified with the `dchost` operand. This operand is applicable only when `dcscddirect=Y` is specified in the TP1/Client/J environment definition. For details, see the `dcscddirect` operand of the TP1/Client/J environment definition.

- `dcscdloadpriority>=Y|N` ~<<`N`>>

  Specifies whether or not the TP1/Server that accepts service requests as a gateway has priority for load balancing.

  `Y`: The TP1/Server that accepts service requests as a gateway has priority for load balancing.

  `N`: The TP1/Server that accepts service requests as a gateway does not have priority for load balancing. TP1/Server balances the workload between nodes.

  This definition is applicable only when RPCs that use the scheduler direct facility are issued (`dcscddirect=Y` is specified in the TP1/Client/J environment definition).

- `dcrapdirect=Y|N` ~<<`Y`>>

  Specifies whether or not the facility for directly querying the RAP-processing listener without querying the information to the TP1/Server's name service (remote API facility) is used.

  To put this operand into effect, specify the port number of the RAP-processing listener in the `dcrapport` operand of the TP1/Client/J environment definition.

  The `dcrapdirect`, `dcscddirect`, and `dcnamuse` operands are mutually exclusive. You cannot specify `Y` for more than one of these operands at the same time; doing so results in a definition error.

- `dcrapport=`*port-number-of-RAP-processing-listener* ~<unsigned integer> ((5001-65535))

  Specifies the port number of the RAP-processing listener of the TP1/Server to which the client belongs or the port number of the firewall. This operand value takes effect on a host for which no port number has been specified with the `dchost` operand.

  This operand is applicable only when `dcrapdirect=Y` is specified in the TP1/Client/J environment definition. For details, see the `dcrapdirect` operand of the TP1/Client/J environment definition.

- `dcrapautoconnect=Y|N` ~<<`N`>>

  Specifies whether or not a permanent connection is established automatically between a CUP and a RAP-processing server.

  `Y`: Establish a permanent connection automatically the next time the method is executed. The target to which a permanent connection establishment request is issued is the RAP-processing listener that is specified in the `dchost` operand of the TP1/Client/J environment definition.

  `N`: Do not establish a permanent connection automatically.

  Operation of various methods based on the connection establishment mode

  Table 5-2 shows connection establishment processing of the non auto connect mode and the auto connect mode.

Table 5-2: Operation of various methods based on the connection establishment mode

Value specified in the dcrapautoconnect operand	openConnection() method is issued	openConnection (host, port) method is issue	openConnection (url, flags) method is issued	rpcCall() is issued
N	• If the dchost operand is specified, a connection is established to the RAP-processing server specified by the dchost operand. • If the dcweburl operand is specified, a connection is established to the URL specified by the dcweburl operand. • If neither the dchost nor dcweburl operand is specified, ErrHostUndefException is returned.	A connection is established to the RAP-processing server specified by the host and port parameters.	A connection is established to the URL specified by the url parameter.	ErrProtoException is returned.

Value specified in the dcrapautoconnect operand	openConnection() method is issued	openConnection (host, port) method is issue	openConnection (url, flags) method is issued	rpcCall() is issued
Y	`ErrProtoException` is returned.	`ErrProtoException` is returned.	`ErrProtoException` is returned.	• If the `dchost` operand is specified, a connection is established to the RAP-processing server specified by the `dchost` operand. • If the `dcweburl` operand is specified, a connection is established to the URL specified by the `dcweburl` operand. • If neither the `dchost` nor `dcweburl` operand is specified, `ErrHostUndefException` is returned.

Notes

- The value of the `dcrapautoconnect` operand can be dynamically changed by the `setRpcextend` method.
- The definition is read by the `rpcOpen` method.
- The definition information read by the `rpcOpen` method is discarded by the `rpcClose` method.
- When an `openConnection` method that specifies a URL as an argument is executed, the specified URL applies only to the `openConnection` method that specified the URL. Once this method has executed, the URL specified in this parameter is not used for another `openConnection` method (which has no parameter specified) that executes after the `closeConnection` method has been used to close the connection.

- `dcerrtrace=Y|N` ~<<N>>

  Specifies whether or not the CUP collects an error trace.

  `Y`: Collect error trace.

  `N`: Do not collect error trace.

  If you are using the CUP as a Java applet, do not specify `Y`; if `Y` is specified, operations cannot be guaranteed.

- `dcerrtracepath=`*directory-for-error-trace-file* ~<character string>

  Specifies the path name of the directory in which error trace files are to be created.

  This operand is applicable only when `dcerrtrace=Y` is specified in the TP1/Client/J environment definition.

- `dcerrtracesize=`*error-trace-file-size* ~<unsigned integer> ((4096-1048576)) <<4096>> (bytes)

  Specifies the size of an error trace file.

- `dcmethodtrace=Y|N` ~<<N>>

  Specifies whether or not the CUP collects a method trace.

  `Y`: Collect method trace.

  `N`: Do not collect method trace.

  If you are using the CUP as a Java applet, do not specify `Y`; if `Y` is specified, operations cannot be guaranteed.

- `dcmethodtracepath=`*directory-for-method-trace-file* ~<character string>

  Specifies the path name of the directory in which method trace files are to be created.

  This operand is applicable only when `dcmethodtrace=Y` is specified in the TP1/Client/J environment definition.

- `dcmethodtracesize=`*method-trace-file-size* ~<unsigned integer> ((4096-1048576)) <<4096>> (bytes)

  Specifies the size of a method trace file.

- `dcuaptrace=Y|N` ~<<N>>

  Specifies whether or not the CUP collects a UAP trace.

  `Y`: Collect UAP trace.

  `N`: Do not collect UAP trace.

  If you are using the CUP as a Java applet, do not specify `Y`; if `Y` is specified, operations cannot be guaranteed.

- `dcuaptracepath=`*directory-for-UAP-trace-file* ~<character string>

  Specifies the path name of the directory in which UAP trace files are to be created.

  This operand is applicable only when `dcuaptrace=Y` is specified in the TP1/Client/J environment definition.

- `dcuaptracesize=`*UAP-trace-file-size* ~<unsigned integer> ((4096-1048576)) <<4096>> (bytes)

  Specifies the size of a UAP trace file.

- `dcdatatrace=Y|N` ~<<`N`>>

  Specifies whether or not the CUP collects a data trace.

  `Y`: Collect data trace.

  `N`: Do not collect data trace.

  If you are using the CUP as a Java applet, do not specify `Y`; if `Y` is specified, operations cannot be guaranteed.

- `dcdatatracepath=`*directory-for-data-trace-file* ~<character string>

  Specifies the path name of the directory in which data trace files are to be created.

  This operand is applicable only when `dcdatatrace=Y` is specified in the TP1/Client/J environment definition.

- `dcdatatracesize=`*data-trace-file-size* ~<unsigned integer> ((4096-1048576)) <<4096>> (bytes)

  Specifies the size of a data trace file.

- `dcdatatracemaxsize=`*maximum-data-size-for-data-trace* ~<unsigned integer> ((16-1048576)) <<128>> (bytes)

  Specifies the maximum size of data per data trace item.

- `dccltrstatisitem=`*statistical-information-item*`[,`*statistical-information-item*`]...`

  Specifies one or more of the character strings listed below to indicate the types of statistical information to be acquired for transaction branches. This operand is applicable only when a transaction is started from the CUP.

  `nothing`

  Do not acquire statistical information.

  `base`

  Acquire the following information as basic information:

- Transaction branch identifier

- Transaction branch determination result

- Transaction branch execution process type

- Name of the server executing a transaction branch

- Name of the service executing a transaction branch

`executiontime`

Acquire the basic information and the transaction branch execution time.

`cputime`

Acquire the basic information and the transaction branch CPU time.

If you specify `nothing`, you cannot specify any other value. If you specify `nothing` together with another value, `nothing` will be ignored.

To acquire statistical information about transactions, specify one of the following settings:

- `trn_tran_statistics=Y` in the transaction service definition
- `-s` option in the `trnstics` command

If you omit this definition, the `trn_statistics_item` operand specification in the RAP-processing listener service definition takes effect.

■ `dccltroptiitem=`*transaction-optimization-item*`[,`*transaction-optimization-item*`]...`

Specifies one or more of the character strings listed below to indicate the optimization items to be used to improve the performance of a global transaction that consists of multiple user servers. This operand is applicable only when a transaction is started from the CUP.

`base`

Optimize the entire synchronization point acquisition processing (prepare, commit, and rollback processing). Because OpenTP1 uses the two-phase commit method for transaction control, four process-to-process communications are required for commit control between two transaction branches.

If all of the following conditions are satisfied, the four process-to-process communications required for commit control are eliminated by having a parent transaction branch execute commit processing for its child transaction branches:

- Both parent and child transaction branches are under the same OpenTP1.

- The parent transaction branch is using a synchronous-response RPC to call its child transaction branches.

- The object for the XA interface of the resource manager that is accessed by a child transaction branch is also linked to its parent transaction branch.

`asyncprepare`

If optimization of the entire synchronization point acquisition processing is not possible because the conditions for `base` are not satisfied, optimize the prepare processing.

If all of the following conditions are satisfied, two of the process-to-process communications are eliminated by executing preparatory processing before an RPC is returned when a child transaction branch issues a service request using the RPC issued by its parent transaction branch:

- Optimization with `base` specified is not available.

- The parent transaction branch is using a synchronous-response RPC to call its child transaction branch.

Note that this optimization increases the response time of a synchronous-response RPC that is issued by the parent transaction branch. For the child transaction branch, the interval from prepare to commit processing increases (because the transaction cannot be determined without an instruction from the parent transaction branch). Therefore, if the OpenTP1 system of the parent transaction branch is shut down and communications are interrupted between the transaction branches, swapping of journal files and validation of the checkpoint dump file are delayed and the OpenTP1 system of the child transaction branch may also be shut down.

You can specify more than one transaction optimization item, in which case the following priority rule applies:
`1 > 2`

`1`: `base`

`2`: `asyncprepare`

If you omit this definition, the `trn_optimum_item` operand specification in the RAP-processing listener service definition takes effect.

■ `dcclttrwatchtime=`*maximum-communication-wait-time-during-transaction-synchronization-point-processing*

~<unsigned integer> ((1~65535)) (seconds)

Specifies the maximum wait time for communications between transaction branches (prepare, commit, or rollback instruction or response) during transaction synchronization point processing. This operand is applicable only when a transaction is started from the CUP.

If no instruction or response is received within the specified time and the corresponding transaction branch is the first phase of a two-phase commit, TP1/Client/J rolls back the transaction branch; if the first phase has already been completed, TP1/Client/J uses a system process of the transaction service to retry transaction determination processing.

When this definition is omitted, the `trn_watch_time` operand in the RAP-processing listener service definition takes effect.

- `dccltrrbinfo=no|self|remote|all`

  Specifies whether or not information about the cause of the rollback is logged when a transaction branch rolls back. This operand is applicable only when a transaction is started from the CUP.

  `no`

  Do not acquire rollback information.

  `self`

  Log rollback information only for those transaction branches that caused rollback.

  `remote`

  Log rollback information not only for those transaction branches applicable to `self` but also for those that received rollback requests from transaction branches in remote nodes.

  `all`

  Log rollback information not only for those transaction branches applicable to `remote` but also for those that received rollback requests from transaction branches in the local node.

  When this definition is omitted, the `trn_rollback_information_put` in the RAP-processing listener service definition takes effect.

- `dccltrlimittime=`*maximum-executable-time-for-transaction-branch*

  ~<unsigned integer> ((0~65535)) (seconds)

  Specifies the maximum executable time for a transaction branch. This operand is applicable only when a transaction is started from the CUP.

  TP1/Client/J sets automatically the timeout values for the `rpcCall` method and synchronization point processing as follows, so that the amount of time from the start of a transaction branch to the end of synchronization point processing does not exceed the time specified in this operand:

- Timeout value for the `rpcCall` method

  If *K* >= *value of this operand*: Timeout occurs and TP1/Client/J returns control with an error without executing the request processing.

  If *K* < *value of this operand* and *(value of this operand* - *K)* >= *W*: TP1/Client/J uses *W* as the timeout value.

  If *K* < *value of this operand* and *(value of this operand* - *K)* < *W*: TP1/Client/J uses *(value of this operand* - *K)* as the timeout value.

  K and *W* indicate the following:

  K: Current time - transaction branch start time

  W: `dcwatchtim` operand value

  >=: Equal to or greater than

- Timeout value for communications within synchronization point processing

  If *K* >= *value of this operand*: TP1/Client/J sets the timeout value to 1 second.

  If *K* < *value of this operand* and *(value of this operand* - *K)* >= *W*: TP1/Client/J uses *W* as the timeout value.

  If *K* < *value of this operand* and *(value of this operand* - *K)* < *W*: TP1/Client/J uses *(value of this operand* - *K)* as the timeout value.

  K and *W* indicate the following:

  K: Current time - transaction branch start time

  W: `dccltttrwatchtime` operand value (if the `dccltttrwatchtime` operand is omitted, the `dcwatchtim` operand value takes effect)

  >=: Equal to or greater than

If time is used for processing other than waiting for data reception, the transaction branch may not be terminated within the specified operand value.

If the time specified in this operand elapses before synchronization point processing begins, the corresponding transaction is rolled back.

When 0 is specified, TP1/Client/J does not monitor the time.

When this definition is omitted, the `trn_limit_time` operand in the RAP-processing listener service definition takes effect.

■ `dcclttrrbrcv=Y|N`

Specifies whether or not a rollback completion report is received after a rollback instruction is sent to the RPC destination transaction branch. This operand is applicable only when a transaction is started from the CUP.

`Y`: Receive a rollback completion report.

`N`: Do not receive a rollback completion report.

When `N` is specified, TP1/Client/J terminates the local transaction branch without receiving a rollback completion report from the RPC destination transaction branch (without waiting for completion of rollback processing on the RPC destination transaction branch).

When this definition is omitted, the `trn_rollback_response_receive` operand in the RAP-processing listener service definition takes effect.

- `dccltrrecoverytype=type1|type2|type3`

Specifies the synchronization point processing method to use in the event of a UAP error. This operand is applicable only when a transaction is started from the CUP.

If an RPC results in a timeout and the address of the target process is unresolved or the UAP executing a transaction is shut down, transaction determination may take time because smooth communications cannot be achieved between transaction branches.

For this operand, select one of the three transaction synchronization point processing methods listed below as appropriate to one of the following types of errors that may occur:

(Error 1) When an RPC results in a timeout:

In this case, the source transaction branch that issued the RPC cannot send a transaction synchronization point message to the RPC destination transaction branch because it does not know which process is executing the service request. As a result, both source and target transaction branches are placed in transaction synchronization point message wait status, requiring time for transaction determination.

(Error 2) When the RAP-processing server shuts down before receiving a response from the RPC:

In this case, the source transaction branch that issued the RPC cannot send a transaction synchronization point message to the RPC destination transaction branch because it does not know which process is executing the service request. As a result, the RPC destination transaction branch is placed in transaction synchronization point message wait status, requiring time for transaction determination.

(Error 3) When the RAP-processing server and the UAP to which the RPC was issued shut down at about the same time after a response from the UAP to which the RPC was issued is received:

In this case, the transaction recovery process that inherited each transaction branch sends a transaction synchronization point message to a nonexistent UAP because it does not know that the remote UAP process has been shut down, requiring time for transaction determination.

The following describes the processing method indicated by each specification value:

`type1`

If (Error 1) occurs, both the source transaction branch that issued the RPC and the RPC destination transaction branch determine transactions when the process for receiving a transaction synchronization point message time out.

If (Error 2) occurs, the source transaction branch that issued the RPC determines the transaction without sending a transaction synchronization point message to the RPC destination transaction branch. The RPC destination transaction branch determines the transaction when the process for receiving a transaction synchronization point message times out.

If (Error 3) occurs, both the source transaction branch that issued the RPC and the RPC destination transaction branch determine transactions when the process for receiving a transaction synchronization point message times out.

`type2`

Same as `type1` if (Error 1) occurs and a transaction is committed.

If (Error 1) occurs and a transaction is rolled back or if (Error 2) occurs, the source transaction branch that issued the RPC sends a transaction synchronization point message to the transaction service process on the node where the RPC destination transaction branch exists, and then determines the transaction. The transaction service process that has received the transaction synchronization point message sends a transaction synchronization point instruction to the process that is processing the applicable transaction branch.

If (Error 3) occurs, both the source transaction branch that issued the RPC and the RPC destination transaction branch determine transactions when the process for receiving a transaction synchronization point message times out.

`type3`

Same as `type1` if (Error 1) occurs and a transaction is committed.

If (Error 1) occurs and a transaction is rolled back or if (Error 2) or (Error 3) occurs, the source transaction branch that issued the RPC sends a transaction synchronization point message to the transaction service process on the node where the remote transaction branch exists. The transaction service process that has received the transaction synchronization point message sends a transaction synchronization point instruction to the process that is processing the applicable transaction branch.

In the following cases, even if you specify `type2` or `type3` in this operand, transaction determination processing may take time:

- The status of the UAP to which the RPC was issued changed during RPC execution (such as an increase in the workload, UAP termination, or UAP shutdown) and a service request was forwarded to the same UAP on another node.
- The version of the remote OpenTP1 server does not support this option.
- The remote transaction branch is taking time for processing other than the transaction synchronization point message reception processing.

When this definition is omitted, the `trn_partial_recovery_type` in the RAP-processing listener service definition takes effect.

■ `dccltrexptm=`*expiry-time-in-transaction-branch*

~<unsigned integer> ((0~65535)) (seconds)

Specifies the maximum transaction branch processing time. This operand is applicable only when a transaction is started from the CUP.

If the transaction branch is not completed within the specified time, TP1/Client/J terminates abnormally the process of that transaction branch and rolls it back. If 0 is specified, TP1/Client/J does not monitor the time.

When this operand is omitted, the `trn_expiration_time` operand in the RAP-processing listener service definition takes effect.

To specify whether or not this value includes the processing time of transaction branches that are executed by other processes when the RPC facility is used, use the `dcclttrexpsp` operand in the TP1/Client/J environment definition.

■ `dccltrcputm=`*transaction-branch-CUP-monitoring-time*

~<unsigned integer> ((0~65535)) (seconds)

Specifies the CPU time allowed for a transaction branch before synchronization point processing starts. This operand is applicable only when a transaction is started from the CUP.

If 0 is specified, TP1/Client/J does not monitor the CUP time.

If the specified time is exceeded, TP1/Client/J terminates abnormally the process of the corresponding transaction branch and rolls it back.

When this operand is omitted, the `trn_cpu_time` operand in the RAP-processing listener service definition takes effect.

■ `dccltrexpsp=Y|N|F`

Specifies whether the monitoring time includes the time of the following types of processing when transaction branch processing is monitored:

1. Time required for a transaction branch being monitored to call another transaction branch using the RPC facility and wait for completion of that processing
2. Time required for a server UAP called by a chained RPC to wait for the next service request
3. Time required for a transaction branch being monitored to execute result reception processing after using a non-response type RPC to call another transaction branch

The following describes each specification value:

`Y`: Include the monitoring time for 1, 2, and 3.

`N`: Include the monitoring time for 3 only.

`F`: Do not include the monitoring time for 1, 2, or 3.

When this operand is omitted, the `trn_expiration_time_suspend` operand in the RAP-processing listener service definition takes effect.

- `dcsndrcvtype=DCCLT_ONEWAY_SND|DCCLT_ONEWAY_RCV|DCCLT_SNDRCV`

  Specifies the environment to initialize when the TCP/IP communication facility is used.

  These operands must be specified for you to use the TCP/IP communication facility.

  `DCCLT_ONEWAY_SND`: Environment for unidirectional sending of messages

  `DCCLT_ONEWAY_RCV`: Environment for unidirectional receiving of messages

  `DCCLT_SNDRCV`: Environment for bidirectional sending and receiving of messages

- `dcrcvport=`*port-number-of-the-receiving-CUP*

  ~<unsigned integer>((1~65535)) <<`11000`>>

  Specifies the port number of the CUP that is receiving the message sent from the MHP.

  Specify the same value as the port number used by the MHP for sending messages. If you are executing several processes and threads concurrently on the same machine, specify different port numbers for each process or thread.

  Do not specify a port number that is being used by the OS or by any other program even if it is available. If such a port number is specified, correct reply data may not be received. Note that different OSs use different port numbers; for details, see the manual for the applicable OS manual.

- `dcsndhost=`*host-name-of-node-on-which-MHP-to-be-connected-exists*~<character string>

  Specifies the host name of the node on which the connection-target MHP exists when a connection is being established for a message to be sent from a CUP.

  You can also specify the host name in the form of a decimal-plus-dot IP address.

- `dcsndport=`*port-number-of-MHP-to-be-connected*

  ~<unsigned integer> ((1~65535)) <<`12000`>>

  Specifies the port number of the connection-target MHP when a connection is being established for a message to be sent from a CUP. Specify a port number different from the one specified in the `dcnamport` operand.

- `dcsockopenatrcv=Y|N` ~<<`N`>>

  Specifies the trigger for opening a reception socket (the trigger to start waiting to receive a connection from a remote computer) when sending and receiving is performed over a single connection while the TCP/IP communication facility is being used.

  If you do not specify this operand, the receive socket is opened when the `rpcOpen` method is executed.

  `Y`: Opens a receive socket if a connection was not established when the `cltReceive` method was executed.

  `N`: Opens a receive socket when the `rpcOpen` method is executed.

- `dcweburl=`*URL-of-TP1/Web-whose-service-is-requested ~<path-name>*

  Specifies the URL of the TP1/Web whose service is being requested.

  Specify the protocol, Web server, the CGI name of the prompter, the TP1/Web service name, and other information in the form of a URL.

  If both the `dchost` operand and the `dcweburl` operand are specified, the `dcweburl` operand takes precedence.

  Note, also, that the `dcscddirect` and `dcnamuse` operands cannot be used if the TP1/Web connection facility is used. If the `dcweburl` operand is defined, and `dcscddirect=Y` and `dcnamuse=Y` are specified, a definition error results.

- `dccltdbgtrcfilecount=`*maximum-number-of-debug-trace-files*

  ~<unsigned integer> ((0~256)) <<`0`>>

  Specifies the maximum number of debug trace files allowed. If `0` is specified, there is no limit to the number of files. If the total number of debug trace files exceeds the value specified in this operand, the file with the oldest update date is deleted.

If multiple TP1Client instances for which different values are specified in the `dccltdbgtrcfilecount` operand are used at the same time, the actual maximum file count cannot be guaranteed.

If you create a TP1Client instance before issuing the `rpcOpen` method, you can also define the `dccltdbgtrcfilecount` operand as a system property. In this case, when the `rpcOpen` method is issued after the TP1Client instance is created, the result obtained by analyzing the content of the `rpcOpen` method is used to overwrite the file count. When the `rpcOpen` method is issued, the specification of the `dccltdbgtrcfilecount` operand defined in the system property becomes invalid.

No error message is issued even when the specification of the `dccltdbgtrcfilecount` operand is invalid. In this case, the default specification for the `dccltdbgtrcfilecount` operand takes effect.

- `dccltrpcmaxmsgsize`=*maximum-size-of-messages-sent-or-received-by-RPC*

~<unsigned integer> ((1~8)) <<`1`>> (MB)

Specifies the maximum size of messages that can be sent or received by RPCs.

If this operand is omitted or if the default (1) is specified, the maximum size of RPC messages that can be sent or received by the `rpcCall` method is 1 MB (1048576 bytes). If the maximum allowable value (8) is specified for this operand, the maximum message size is 8 MB (8388608 bytes). You cannot specify a send or receive message size that exceeds the value specified in this operand as an argument of the `rpcCall` method. If the specified send or receive message size exceeds the value specified in this operand, the `rpcCall` method returns `ErrInvalidArgsException`.

When you specify the `dccltrpcmaxmsgsize` operand, do not use the facilities listed below. Otherwise, an error occurs at the communication target TP1/Server node.

- RPC that uses the scheduler direct facility
- RPC that specifies the communication target
- TP1/Web connection facility

The following table shows what happens when a service request is issued to a TP1/Server other than a TP1/Server (version 06-02 or later) that supports the `rpc_max_message_size` operand of the system common definition:

Table 5-3: Action of the rpcCall method when 2 or more is specified in the dccltrpcmaxmsgsize operand

Method	Definition that specifies the RPC type	Version of TP1/ Server that acts as gateway	Node version[#1]	
			Before 06-02	06-02 or later[#2]
`rpcCall`	`dcrapdirect=Y`	Before 06-02	N[#3]	N[#3]
		06-02 or later	N[#4]	Y
	`dcnamuse=Y`	Before 06-02	N[#4]	Y
		06-02 or later	N[#4]	Y

Legend:

Y: Can be requested.

N: Cannot be requested.

#1

Version of the node where the SPP is running and that was defined in the all_node operand specified for the TP1/Server that acts as a gateway.

#2

When the rpc_max_message_size operand is specified in the system common definition.

#3

When the input data exceeds 1 MB, `ErrMessageTooBigException` occurs; when the input data is 1 MB or smaller and if the output area size exceeds 1 MB, `ErrInvalidArgsException` occurs.

#4

`ErrNoSuchServiceGroupException` occurs.

■ `dcscdhostchange=Y|N ~<<N>>`

Specifies whether or not the service request target schedulers specified in the `dchost` operand of the TP1/Client/J environment definition are distributed to the RPCs.

`Y`: Distributes the service request target schedulers to the RPCs.

`N`: Does not distribute the service request target schedulers to the RPCs.

When `Y` is specified, the following processing occurs:

For the first service required by an RPC, if `dchostselect=Y` is specified in the TP1/Client/J environment definition, a service request target scheduler specified in the `dchost` operand of the TP1/Client/J environment definition is randomly selected. If `dchostselect=N` is specified in the TP1/Client/J environment definition, the first service request target scheduler specified in the `dchost` operand of the TP1/Client/J environment definition is selected.

For the second and subsequent RPC service requests, the service request target schedulers are selected in the round-robin mode according to the order specified in the `dchost` operand, beginning with the service request target scheduler previously selected.

- `dccltloadbalance=Y|N ~<<N>>`

  Specifies whether or not to store in cache the information on the service request target scheduler with the smallest workload from the information on multiple service request target schedulers obtained from the name server.

  `Y`: Stores the information on the service request target scheduler with the smallest workload in the cache.

  `N`: Does not store the information on the service request target scheduler with the smallest workload in the cache.

  When `Y` is specified, the following processing occurs:

  If multiple service request target schedulers are stored in the cache, the first service request target scheduler is randomly selected. For the second and subsequent RPC service requests, the service request target schedulers stored in the cache are selected in the round-robin mode.

  For details about workload levels, see the manual *OpenTP1 Description*.

- `dccltcachetim=`*cache-expiration-time*

  ~<unsigned integer> ((0~65535)) <<`30`>> (seconds)

  Specifies how long information on a service request target scheduler stored in the cache remains valid. Information on a service request target scheduler that has reached the cache expiration time is deleted from the cache. The cache is updated when information on the service request target scheduler is subsequently acquired again and stored in the cache.

  This operand is valid only when `dccltloadbalance=Y` is specified in the TP1/Client/J environment definition.

- `dccltconnecttimeout=`*maximum-time-for-monitoring-connection-establishment*

  ~<unsigned integer> ((0~65535)) <<`0`>> (seconds)

  Specifies the maximum amount of monitoring time it takes to establish a connection during data transmission.

The value specified in this operand is not the processing time specified for the `java.net.Socket.connect` method (hereafter referred to as the `connect` method), but is the maximum amount of monitoring time it takes for the `connect` method to establish a connection.

If a connection cannot be established because the remote system has not been started, for example, the method issued by a CUP may return an error before the monitoring time specified in this operand elapses. This is because the connection establishment monitoring time by the OS takes precedence over the maximum monitoring time specified in this operand. The connection establishment monitoring time by the OS, the number of times connection establishment requests can be resent, and the resend interval vary depending on the platform. Furthermore, depending on the method or function used, the processing time of the `connect` method may exceed the value specified in this operand.

If `0` is specified or this operand is omitted, the OS monitors the connection establishment process.

■ `dccltprftrace=Y|N ~<<Y>>`

Specifies whether or not a performance analysis trace of the Cosminexus Application Server is collected when TP1/Client/J is run on a Cosminexus Application Server.

`Y`: Collects a performance analysis trace. However, no performance analysis trace is collected if the PRF daemon has not been started.

`N`: Does not collect a performance analysis trace.

■ `dccltprfinfosend=Y|N ~<<Y>>`

Specifies whether or not identification information in TP1/Client/J is added to the information output to the performance verification trace of OpenTP1 when an RPC that uses the name service or scheduler direct facility is issued to TP1/Server.

`Y`: Adds identification information inside TP1/Client/J.

`N`: Does not add identification information inside TP1/Client/J.

■ `dccltnammlthost=Y|N ~<<N>>`

Specifies whether or not an RPC that uses the name service is issued to a TP1/Server in the multi-homed host environment.

`Y`: Issues an RPC that uses the name service to a TP1/Server in the multi-homed host environment. Specify this option when multiple RPC request targets are defined or when there are more than one TP1/Servers in the multi-homed host environment.

`N`: Does not issue an RPC that uses the name service to a TP1/Server in the multi-homed host environment. An error may occur if an RPC that uses the name service is issued to a TP1/Server in the multi-homed host environment.

For details, see *2.2.9(3) Definition for issuing an RPC to TP1/Server in the multi-homed host environment*.

- `dccltdatacomp=Y|N` ~ <<N>>

  Specifies whether or not the data compression facility is used.

  `Y`: Uses the data compression facility.

  `N`: Does not use the data compression facility.

- `dccltconnectinf=`*terminal-identification-information*

  Specifies (in EBCDIK codes) the logical terminal name of the DCCM3 logical terminal as the terminal identification information. The terminal identification information is expressed as the value `0x` followed by a maximum of 128 hexadecimal characters (`0x` at the beginning is not included in the 128 characters). You can specify a maximum of 64 bytes (128 characters), where each byte consists of 2 characters. Note that DCCM3 uses only the first 8 bytes and ignores the remainder of the value.

  You can change this operand value dynamically by calling the `setConnectInformation` method. Once the `setConnectInformation` method has been called, the value specified in this operand is ignored until the `rpcOpen` method is called again.

  The value specified in this operand takes effect when DCCM3 logical terminal host name is specified in the `dchost` operand and the port number of the DCCM3 logical terminal is specified in the `dchost` or `dcrapport` operand in the TP1/Client/J environment definition, and a permanent connection is established with the DCCM3 logical terminal by means of one of the following methods:

  - Calling the `openConnection` method. In the case of the `openConnection` method with parameters, specify DCCM3 logical terminal host name in the `host` parameter and the port number of the DCCM3 logical terminal in the `port` parameter.
  - Specifying `Y` in the `dcrapautoconnect` operand in the TP1/Client/J environment definition and then call the `rpcCall` method.

  If this operand is omitted, terminal identification information is not sent to the DCCM3 logical terminal. However, if the `setConnectInformation` method is called, the terminal identification information specified in the `setConnectInformation` method is sent to the DCCM3 logical terminal when a permanent connection with the DCCM3 logical terminal is established.

- `dcscdmulti=Y|N ~` <<N>>

Specifies whether or not the multi-scheduler facility is to be used.

`Y`: Uses the multi-scheduler facility.

`N`: Does not use the multi-scheduler facility.

When you use the multi-scheduler facility, you can reduce the workload of scheduling by randomly selecting one of the multi-scheduler daemons that have been started.

If you specify `Y` in this definition, also pay attention to the `dchost`, `dcscdport`, and `dcscdmulticount` TP1/Client/J environment definitions. Note that this definition is not applicable during execution of the `rpcCallTo` method.

- `dcscdmulticount=`*processes-count-for-multi-scheduler-daemon* ~ <unsigned integer> ((1-4096)) <<1>>

  Specifies the number of processes for the multi-scheduler daemon. This value must be equal to or less than the number of processes specified in the `-m` option of the `scdmulti` schedule service definition.

  This definition is applicable when `dcscdmulti=Y` and `dcscddirect=Y` are specified in the TP1/Client/J environment definition. In such a case, the port number is selected randomly from within the following range of values:

  - Minimum value: Port number specified in `dchost` or `dcscdport`
  - Maximum value: Minimum value + number of processes specified in `dcscdmulticount` - 1

  If the maximum value for this port number exceeds 65535, `ErrFatalException` occurs during definition analysis and `ErrInvalidPortException` occurs during `setDchost` processing.

  If the specified value is greater than the number of processes specified in the `-m` option of `scdmulti`, an attempt may be made to establish connection with a port where the multi-scheduler is not running, resulting in the `ErrNetDownAtClientException`.

- `dccltcupsndhost=`*transmission-source-host* ~ <character string>

  Specifies the transmission source host during the following connection establishments:

  - RPC using the scheduler direct facility
  - RPC using the name service
  - RPC using the remote API facility
  - RPC specifying the communication target
  - TCP/IP communication facility

For the host name, you can also specify an IP address in decimal dot notation.

An exception occurs on the issued method in the following cases:

- `localhost` was specified as the host name
- The specified IP address begins with `127`
- The specified host does not exist on the system where the CUP is being executed

If this definition is omitted, the transmission source host is allocated arbitrarily.

- `dccltcuprcvport=`*port-number-used-for-reception-of-CUP* ~ <unsigned integer> ((5001-65535))

Specifies the port number of the CUP that is to receive messages from the server.

The port number specified in this definition takes effect when the following functions are used:

- When an RPC using the scheduler direct facility is received (receive port)
- When an RPC using the name service is received (receive port)

If this definition is omitted, a port number allocated arbitrarily by the system is used.

If you execute multiple processes or threads concurrently within the same machine, specify different port numbers. If a duplicated port number is specified, correct reply data may not be received. Note that different OSs use different port number; for details, see the manual for the applicable OS.

- `dcclttrcmplmttm=`*maximum-time-allowed-for-completion-of-transaction* ~ <unsigned integer> ((0-65535)) (seconds)

Specifies the maximum execution time from startup to termination of a transaction branch that is executed by the RAP-processing server. If the specified time is exceeded, the transaction branch settles to either rollback or commit by the recovery process, and then is terminated after the RAP-processing server processing terminates abnormally.

If `0` is specified, the maximum transaction branch execution time is not monitored.

If this specification is omitted, processing depends on the `trn_completion_limit_time` operand specification in the RAP-processing listener service definition.

If this facility is used with a TP1/Server version earlier than 07-01, the facility runs in invalid status.

5.2.3 Notes about specifying the TP1/Client/J environment definition

- Do not specify any characters other than the settings on any of the lines containing TP1/Client/J environment definition operands. If a specified character is not a valid part of a setting, the `rpcOpen` method may return `ErrFatalException`.
- TP1/Client/J assumes that a line beginning with a TP1/Client/J environment definition operand (spaces and tabs at the beginning of lines are ignored) is subject to definition analysis. TP1/Client/J regards all lines containing a character string that is not a valid TP1/Client/J environment definition operand as comment lines.

Chapter
6. Error Handling

This chapter describes how to handle errors.

6.1 Collecting trace information
6.2 Handling network errors
6.3 Validity of timer values
6.4 Handling other errors

6.1 Collecting trace information

The troubleshooting facility of TP1/Client/J allows you to collect the following types of trace information:

- UAP traces
- Data traces
- Error traces
- Memory traces
- Method traces
- Debug traces
- Performance analysis traces
- Performance verification traces

The troubleshooting facility of TP1/Client/J enables you to collect UAP, data, error, method, debug, and memory traces. To simplify troubleshooting, we recommend that you collect trace information. In the event of an error, you can determine the cause of the error from the trace information. If an error occurs but trace information has not been collected, set the system to collect traces, repeat the event that resulted in the error, and then refer to the collected trace information.

For details about trace information, see *2.11 Troubleshooting facility*.

6.2 Handling network errors

In a JDK 5.0 environment, it is not possible to collect adequate error information for assistance in the event of a network error. We recommend that you use a network in a non-Java environment to check for the cause of an error.

6.3 Validity of timer values

In a system that uses TCP/IP for communication, you cannot detect errors in real time due to TCP/IP limitations. Therefore, a time-monitoring timeout triggers the system to assume that an error has been detected. The OpenTP1 system enables you to specify timer monitoring for each type of processing. You should set an optimum value for each setting.

If an error such as system shutdown occurs in the RAP-processing server system while connection with the RAP-processing server is being established, Java cannot detect the disconnection. In such a case, TP1/Client/J waits until the response wait time specified with the `setDcwatchtim` method is reached, then returns an error with the exception `ErrTimedOutException`.

6.4 Handling other errors

Errors may result from malfunctions in the RAP-processing server in the TP1/Server system or in the target TP1/Server whose service is requested or from invalid definition values. In such cases, you should check such information as the TP1/Server's log files to determine the cause.

Appendix

A. Changes During Upgrading

The changes between versions are categorized as follows:

- Additions/deletions of APIs, definitions, and commands
- Changes to operation
- Changes in the default values for APIs, definitions, and commands

A.1 Changes in 07-02

The following table shows the additions and deletions that were made to APIs, definitions, and commands in TP1/Client/J 07-02.

Table A-1: Additions/deletions to APIs, definitions, and commands in TP1/Client/J 07-02

Classification	Category	Description
Addition	API	`TP1Client` class • `acceptNotification` method • `cancelNotification` method • `openNotification` method • `acceptNotificationChained` method • `closeNotification` method
		`ErrVersionException` class
		`ErrAcceptCanceledException` class
	Definition	TP1/Client/J environment definition • `dcscdmulti` operand • `dcscdmulticount` operand • `dccltcupsndhost` operand • `dccltcuprcvport` operand • `dcclttrcmplmttm` operand
	Command	None
Deletion	None	

The following table shows the changes to operation in TP1/Client/J 07-02.

Table A-2: Changes to operation in TP1/Client/J 07-02

Category	Description
API	A change was made so that UAP traces are acquired by the `setUapTraceMode`, `setErrorTraceMode`, `setMethodTraceMode`, and `setDataTraceMode` methods.

Category	Description
Definition	None
Command	None
Other	A change was made so that identification information unique to each instance of TP1/Client/J is output to the TP1/Server performance verification trace by the PRC using the remote API facility.
	A change was made so that identification information unique to each instance of TP1/Client/J is output to UAP traces.

There were no changes in the default values for APIs, definitions, and commands in TP1/Client/J 07-02.

A.2 Changes in 07-01

The following table shows the additions and deletions that were made to APIs, definitions, and commands in TP1/Client/J 07-01.

Table A-3: Additions/deletions to APIs, definitions, and commands in TP1/Client/J 07-01

Classification	Category	Description
Addition	API	`TP1Client` class • `setConnectInformation` method
	Definition	TP1/Client/J environment definition • `dccltdatacomp` operand • `dccltconnectinf` operand
	Command	None
Deletion	None	

The following table shows the changes to operation in TP1/Client/J 07-01.

Table A-4: Changes to operation in TP1/Client/J 07-01

Category	Description
API	None
Definition	None
Command	None

Category	Description
Other	A change was made so that retry processing is performed when an error (`java.net.BindException` exception) occurs due to an insufficient number of ports having been allocated automatically in the local system when a connection is established from TP1/Client/J to TP1/Server.
	A correction was made to a problem with the `ErrClientTimedOutException` exception that resulted when a temporary close request message was received from SPP during reply message reception processing, because the temporary close request message was discarded in order to receive the original reply message, and then reception was further attempted illegally.
	A correction was made so that an invalid `ErrRMErrException` exception is not thrown from TP1/Client/J when the XA resource service facility is used.

There were no changes in the default values for APIs, definitions, and commands in TP1/Client/J 07-01.

A.3 Changes in 07-00

The following table shows the additions and deletions that were made to APIs, definitions, and commands in TP1/Client/J 07-00.

Table A-5: Additions/deletions to APIs, definitions, and commands in TP1/Client/J 07-00

Classification	Category	Description
Addition	API	None
	Definition	TP1/Client/J environment definition • `dccltnammlthost` operand
	Command	None
Deletion	None	

The following table shows the changes to operation in TP1/Client/J 07-00.

Table A-6: Changes to operation in TP1/Client/J 07-00

Category	Description
API	None
Definition	None
Command	None
Other	The JDK versions supported by TP1/Client/J were changed to Java(TM)2 Software Development Kit, Standard Edition, and Version 5.0 (JDK 5.0) or later.

There were no changes in the default values for APIs, definitions, and commands in TP1/Client/J 07-00.

Index

A

abbreviations defined iii
acceptNotification 213
acceptNotification method, information for 99
acceptNotificationChained 218
acceptNotificationChained method, information for 101
API
    how to use 144
    list of 140
    using connect method 66
API, execution order of 144
    calling remote service (RPC) 146
    connecting to TP1/Server's RAP-processing server 146
    creating instance of TP1Client class 144
    initializing RPC environment 145
    preparing to use TP1/Client/J 144
    releasing permanent connection with RAP-processing listener or server 147
    releasing RPC environment 148
assembly facility, received message 52
auto connect mode 12

C

cache-expiration-time 310
cancelNotification 216
cancelNotification method, information for 100
chained RPC 17
closeConnection 171
closeConnection method, information for 90
closeNotification 221
cltAssemReceive 209
cltAssemReceive method, information for 98
cltAssemSend 207
cltAssemSend method, information for 97
cltReceive 202
cltReceive method, information for 96
cltSend 205
cltSend method, information for 97
code
    correspondence to exception name 112
    correspondence to method name 108
commit 37
    in chained mode 37
    in unchained mode 37
connect method 66
    API using 66
    exceptions when connect method times out 67
connect mode
    auto 12
    non-auto 12
conventions
    abbreviations iii
    diagrams vi
    fonts and symbols vi
    KB, MB, GB, and TB viii
    version numbers viii
Cosminexus Application Server
    collating, with OpenTP1 trace 116
    outputting performance analysis trace on 115

D

data compression facility 128, 312
    effects of 129
data trace 105
    collecting 298
data-trace-file-size 298
dccache 292
dccltcachetim 310
dccltconnectinf 312
dccltconnecttimeout 310
dccltcuprcvport 314
dccltcupsndhost 313
dccltdatacomp 312
dccltdbgtrcfilecount 307

dccltdelay 291
dccltextend 292
dccltinquiretime 290
dccltloadbalance 310
dccltnammlthost 311
dccltprfinfosend 311
dccltprftrace 311
dccltrpcmaxmsgsize 308
dcclttrcmplmttm 314
dcclttrcputm 305
dcclttrexpsp 305
dcclttrexptm 305
dcclttrlimittime 301
dcclttroptiitem 299
dcclttrrbinfo 301
dcclttrrbrcv 302
dcclttrrecoverytype 303
dcclttrstatisitem 298
dcclttrwatchtime 300
DCCM3
    connection facility 70
    issuing ROC to 70
    TCP/IP communication with 75
DCCM3 logical terminal, sending terminal identification information to 75
dcdatatrace 298
dcdatatracemaxsize 298
dcdatatracepath 298
dcdatatracesize 298
dcerrtrace 297
dcerrtracepath 297
dcerrtracesize 297
dchost 289
dchostselect 292
dcmethodtrace 297
dcmethodtracepath 297
dcmethodtracesize 297
dcnamport 289
dcnamuse 288
dcrapautoconnect 294
dcrapdirect 294
dcrapport 294
dcrcvport 306
DCRpcBindTbl 223
dcscddirect 293
dcscdhostchange 309
dcscdloadpriority 293
dcscdmulti 312
dcscdmulticount 313
dcscdport 293
dcselint 292
dcsndhost 307
dcsndport 307
dcsndrcvtype 306
dcsockopenatrcv 307
dcuaptrace 297
dcuaptracepath 298
dcuaptracesize 298
dcwatchtim 290
dcwatchtiminherit 291
dcwatchtimrpcinherit 291
dcweburl 307
debug trace 114
definition 279
    overview of 280
    rules for 284
diagram conventions vi
directory-for-data-trace-file 298
directory-for-error-trace-file 297
directory-for-method-trace-file 297
directory-for-UAP-trace-file 298
dynamic definition changing facility 65

E

ErrAcceptCanceledException 224
ErrBufferOverflowException 225
ErrClientTimedOutException 227
ErrCollisionMessageException 226
ErrConnfreeException 228
ErrConnRefusedException 229
ErrFatalException 230
ErrHazardException 231
ErrHazardNoBeginException 232
ErrHeuristicException 233
ErrHeuristicNoBeginException 234
ErrHostUndefException 235
ErrInitializingException 237
ErrInvalidArgsException 238

ErrInvalidMessageException 239
ErrInvalidPortException 240
ErrInvalidReplyException 242
ErrIOErrException 243
ErrMessageTooBigException 244
ErrNetDownAtClientException 245
ErrNetDownAtServerException 246
ErrNetDownException 247
ErrNoBeginException 248
ErrNoBufsAtServerException 249
ErrNoBufsException 250
ErrNoSuchServiceException 251
ErrNoSuchServiceGroupException 252
ErrNotTrnExtendException 253
ErrNotUpException 254
error handling 317
error information, collecting 149
error trace 105
    collecting 297
error-trace-file-size 297
ErrProtoException 255
ErrReplyTooBigException 256
ErrRMException 257
ErrRollbackException 258
ErrRollbackNoBeginException 259
ErrSecchkException 260
ErrSecurityException 261
ErrServerBusyException 262
ErrServerTimedOutException 263
ErrServiceClosedException 264
ErrServiceNotUpException 265
ErrServiceTerminatedException 266
ErrServiceTerminatingException 267
ErrSyserrAtServerException 268
ErrSyserrException 269
ErrTestmodeException 270
ErrTimedOutException 271
ErrTMException 273
ErrTrnchkException 274
ErrTrnchkExtendException 275
ErrVersionException 276
exception name, correspondence to code 112
expiry-time-in-transaction-branch 305

F

facility extension level, setting 292
facility, adjusting 149
file access authority, specifying 9
firewall 19
font conventions vi
function for allocating a fixed terminal 76

G

GB meaning viii
getTrnID 201
getTrnID method, information for 95

H

heuristic situation, handling of 40
host-name-of-node-on-which-MHP-exists 307
host-name-of-TP1/Server-as-a-gateway 289

I

identifiers of current transaction, acquiring 42
important note on this manual viii
inter-node load-balancing facility 18

K

KB meaning viii

M

master scheduler daemon 31
maximum-communication-delay-time 291
maximum-communication-wait-time-during-transaction-synchronization-point-processing 300
maximum-data-size-for-data-trace 298
maximum-entries-count-in-name-cache 292
maximum-executable-time-for-transaction-branch 301
maximum-number-of-debug-trace-files 307
maximum-permanent-connection-inquiry-interval 290
maximum-size-of-messages-sent-or-received-by-RPC 308
maximum-time-allowed-for-completion-of-transaction 314

maximum-time-for-monitoring-connection-establishment 310
maximum-time-to-wait-for-a-response 290
MB meaning viii
memory trace 105
message
  bidirectional receiving 50
  bidirectional sending 50
  notes on receiving of 53
  notes on sending of 53
  unidirectional receiving of 47
  unidirectional sending of 46
method name, correspondence to code 108
method trace 114
  collecting 297
method-trace-file-size 297
multi-scheduler daemon 31
multi-scheduler facility 31, 313

N

network error, handling 319
non-auto connect mode 12
non-response type RPC 16
  relationship with synchronization point 41

O

openConnection 166, 168, 170
openConnection() method, information for 84
openConnection(host, port) method, information for 84
openConnection(url, flags) method, information for 86
openNotification 217
openNotification method, information for 100
OpenTP1 client
  feature of 2
  functionality 11
  overview of 1
OpenTP1, load-balancing facility 18
operand 288

P

performance analysis trace 115
  collecting, of Cosminexus Application Server 311
  collection point 117
  example of outputting 116
  information collected in 116
  outputting, on Cosminexus Application Server 115
performance verification trace 116
  adding identification information inside TP1/Client/J as information output to 311
  identification information is added to 117
permanent connection 12
  automatically establishing 294
  establishing 12
  releasing 12
port-number-of-MHP-to-be-connected 307
port-number-of-name-service 289
port-number-of-RAP-processing-listener 294
port-number-of-schedule-service 293
port-number-of-the-receiving-CUP 306
port-number-used-for-reception-of-CUP 314
PRF trace 115
processes-count-for-multi-scheduler-daemon 313
program interface 139

R

RAP-processing listener, releasing permanent connection with 147
RAP-processing server
  connecting to TP1/Server's 146
  releasing permanent connection with 147
  whether or not RAP-processing server is to inherit CUP's maximum time to wait for response 291
receive port fixing facility 133
received message assembly facility 52
reception socket, trigger for opening 307
remote API facility 19, 294
rollback 38
  completion report, receiving 302
  in chained mode 39
  in unchained mode 39
  logging information about cause of 301
RPC 15

- calling 146
- chained 17
- issuing RPC using name service to TP1/Server in multi-homed host environment 311
- issuing, to TP1/Server in multi-homed host environment 25
- non-response type 16
- synchronous-response 16
- time monitoring of 19
- using multi-scheduler facility 31
- using name service 22, 288
- using remote API facility 19
- using scheduler direct facility 21, 293
- using ServerSocket for 30
- with communication destination specified 27

RPC environment
- initializing 145
- releasing 148

rpcCall 172, 180
rpcCall method, information for 86
rpcCallTo 188
rpcCallTo method, information for 88
rpcClose 166
rpcOpen 164, 165
rpcOpen method, information for 93

S

scheduler direct facility 21
scheduling facility 18
service request target scheduler
- distributing to RPCs 309
- storing in cache information on, with smallest workload 310

setConnectInformation 212
setConnectInformation method, information for 98
setDataTraceMode 162
setDataTraceMode method, information for 103
setDccltdelay 156
setDccltdelay method, information for 91
setDccltextend 158
setDccltextend method, information for 91
setDccltinquiretime 155
setDccltinquiretime method, information for 91
setDchost 159
setDchost method, information for 92
setDcselint 157
setDcselint method, information for 91
setDcwatchtim 157
setDcwatchtim method, information for 90
setErrorTraceMode 163
setErrorTraceMode method, information for 102
setMethodTraceMode 161
setMethodTraceMode method, information for 102
setRpcextend 158
setRpcextend method, information for 92
setScdDirectObject 191
setTraceArray 160
setUapTraceMode 160
setUapTraceMode method, information for 101
source host specification facility 131
SPP 2
statistical-information-item 298
symbol conventions vi
synchronization point acquisition 37
synchronization point processing, method to be used in the event of a UAP error 303
synchronous-response RPC 16
- relationship with synchronization point 41
- timeout, reduction of server workload in event of 29

syntax conventions vii

T

TB meaning viii
TCP/IP communication facility 46
- environment to initialize when being used 306

TCP/IP connection, facility for monitoring establishment 66
terminal identification information setting facility 76
terminal-identification-information 312
time monitoring 18, 19
timer value, validity of 320
TP1/Client/J
- adjustment during execution 148
- classes used with 153
- development environment 9
- environment definition 287

environment definitions, list of 280
environment of 8
execution environment 10
installation 8
operation, mechanism of 5
preparing to use 144
relationship with TP1/Server 2

TP1/Server
as gateway 5
as gateway, having priority for load balancing 293
as gateway, selecting randomly 292
connecting to RAP-processing server 146
in multi-homed host environment, issuing RPC to 25
in multi-homed host environment, issuing RPC using name service to 311
inheriting CUP's maximum time to wait for response 291
relationship with TP1/Client/J 2
scheduling facility of 18

TP1/Web
connecting to (starting session) 59
connection facility 59
releasing connection (ending session) 60
RPC facility for connecting to 60
service requests to 59

TP1Client 154
TP1Client class, creating instance of 144
TP1ClientException 277
trace information, collecting 318
trace output, instructing 150
transaction
reporting information about current 42
verifying synchronization point in event of error 42

transaction branch processing, monitoring 306
transaction branch, type of statistical information to be acquired for 298
transaction control 35
transaction-branch-CUP-monitoring-interval 305
transaction-optimization-item 299
transmission-source-host 313
trnBegin 192
trnBegin method, information for 93
trnChainedCommit 193
trnChainedCommit method, information for 104
trnChainedRollback 196
trnChainedRollback method, information for 104
trnInfo 201
trnInfo method, information for 95
trnUnchainedCommit 198
trnUnchainedCommit method, information for 104
trnUnchainedRollback 199
trnUnchainedRollback method, information for 104
troubleshooting facility 82

U

UAP trace 83
collecting 297

UAP-trace-file-size 298
unidirectional server message reception facility 55
URL-of-TP1/Web-whose-service-is-requested 307

V

version number conventions viii

X

XA resource service facility 80

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)