HITACHI
Inspire the Next

OpenTP1 Version 7

Programming Reference C Language

3000-3-D54-30(E)

■ **Relevant program products**

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

For AIX 5L V5.1, AIX 5L V5.2, AIX 5L V5.3, and AIX V6.1

P-1M64-2131 uCosminexus TP1/Server Base 07-03*

P-1M64-2331 uCosminexus TP1/FS/Direct Access 07-03*

P-1M64-2431 uCosminexus TP1/FS/Table Access 07-03*

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

P-1M64-2631 uCosminexus TP1/Offline Tester 07-00

P-1M64-2731 uCosminexus TP1/Online Tester 07-00

P-1M64-2831 uCosminexus TP1/Multi 07-00

P-1M64-2931 uCosminexus TP1/High Availability 07-00

P-1M64-3131 uCosminexus TP1/Message Control 07-03

P-1M64-3231 uCosminexus TP1/NET/Library 07-04

P-1M64-8131 uCosminexus TP1/Shared Table Access 07-00

P-1M64-8331 uCosminexus TP1/Resource Manager Monitor 07-00

P-1M64-8531 uCosminexus TP1/Extension 1 07-00

P-1M64-C371 uCosminexus TP1/Message Queue 07-01

P-1M64-C771 uCosminexus TP1/Message Queue - Access 07-01

P-F1M64-31311 uCosminexus TP1/Message Control/Tester 07-00

P-F1M64-32311 uCosminexus TP1/NET/User Agent 07-00

P-F1M64-32312 uCosminexus TP1/NET/HDLC 07-00

P-F1M64-32313 uCosminexus TP1/NET/X25 07-00

P-F1M64-32314 uCosminexus TP1/NET/OSI-TP 07-00

P-F1M64-32315 uCosminexus TP1/NET/XMAP3 07-01

P-F1M64-32316 uCosminexus TP1/NET/HSC 07-00

P-F1M64-32317 uCosminexus TP1/NET/NCSB 07-00

P-F1M64-32318 uCosminexus TP1/NET/OSAS-NIF 07-01

P-F1M64-3231B uCosminexus TP1/NET/Secondary Logical Unit - TypeP2 07-00

P-F1M64-3231C uCosminexus TP1/NET/TCP/IP 07-02

P-F1M64-3231D uCosminexus TP1/NET/High Availability 07-00

P-F1M64-3231U uCosminexus TP1/NET/User Datagram Protocol 07-00

R-1M45F-31 uCosminexus TP1/Web 07-00

For AIX 5L V5.3 and AIX V6.1

P-1M64-1111 uCosminexus TP1/Server Base(64) 07-03*

P-1M64-1311 uCosminexus TP1/FS/Direct Access(64) 07-03*

P-1M64-1411 uCosminexus TP1/FS/Table Access(64) 07-03*

P-1M64-1911 uCosminexus TP1/High Availability(64) 07-00

P-1M64-1L11 uCosminexus TP1/Extension 1(64) 07-00

For HP-UX 11i V1 (PA-RISC) and HP-UX 11i V2 (PA-RISC)

P-1B64-3F31 uCosminexus TP1/NET/High Availability 07-00

P-1B64-8531 uCosminexus TP1/Extension 1 07-00

P-1B64-8931 uCosminexus TP1/High Availability 07-00

R-18451-41K uCosminexus TP1/Client/W 07-00

R-18452-41K uCosminexus TP1/Server Base 07-00

R-18453-41K uCosminexus TP1/FS/Direct Access 07-00
R-18454-41K uCosminexus TP1/FS/Table Access 07-00
R-18455-41K uCosminexus TP1/Message Control 07-03*
R-18456-41K uCosminexus TP1/NET/Library 07-04*
R-18459-41K uCosminexus TP1/Offline Tester 07-00
R-1845A-41K uCosminexus TP1/Online Tester 07-00
R-1845C-41K uCosminexus TP1/Shared Table Access 07-00
R-1845D-41K uCosminexus TP1/Resource Manager Monitor 07-00
R-1845E-41K uCosminexus TP1/Multi 07-00
R-1845F-41K uCosminexus TP1/Web 07-00
R-F18455-411K uCosminexus TP1/Message Control/Tester 07-00
R-F18456-411K uCosminexus TP1/NET/User Agent 07-00
R-F18456-415K uCosminexus TP1/NET/XMAP3 07-01*
R-F18456-41CK uCosminexus TP1/NET/TCP/IP 07-02*
For HP-UX 11i V2 (IPF) and HP-UX 11i V3 (IPF)
P-1J64-3F21 uCosminexus TP1/NET/High Availability 07-00
P-1J64-4F11 uCosminexus TP1/NET/High Availability(64) 07-00
P-1J64-8521 uCosminexus TP1/Extension 1 07-00
P-1J64-8611 uCosminexus TP1/Extension 1(64) 07-00
P-1J64-8921 uCosminexus TP1/High Availability 07-00
P-1J64-8A11 uCosminexus TP1/High Availability(64) 07-00
P-1J64-C371 uCosminexus TP1/Message Queue 07-01
P-1J64-C571 uCosminexus TP1/Message Queue(64) 07-01
P-1J64-C871 uCosminexus TP1/Message Queue - Access(64) 07-00
R-18451-21J uCosminexus TP1/Client/W 07-02
R-18452-21J uCosminexus TP1/Server Base 07-03*
R-18453-21J uCosminexus TP1/FS/Direct Access 07-03*
R-18454-21J uCosminexus TP1/FS/Table Access 07-03*
R-18455-21J uCosminexus TP1/Message Control 07-03*
R-18456-21J uCosminexus TP1/NET/Library 07-04*
R-18459-21J uCosminexus TP1/Offline Tester 07-00
R-1845A-21J uCosminexus TP1/Online Tester 07-00
R-1845C-21J uCosminexus TP1/Shared Table Access 07-00
R-1845D-21J uCosminexus TP1/Resource Manager Monitor 07-00
R-1845E-21J uCosminexus TP1/Multi 07-00
R-1845F-21J uCosminexus TP1/Web 07-00
R-1B451-11J uCosminexus TP1/Client/W(64) 07-02
R-1B452-11J uCosminexus TP1/Server Base(64) 07-03*
R-1B453-11J uCosminexus TP1/FS/Direct Access(64) 07-03*
R-1B454-11J uCosminexus TP1/FS/Table Access(64) 07-03*
R-1B455-11J uCosminexus TP1/Message Control(64) 07-03*
R-1B456-11J uCosminexus TP1/NET/Library(64) 07-04*
R-F18455-211J uCosminexus TP1/Message Control/Tester 07-00
R-F18456-215J uCosminexus TP1/NET/XMAP3 07-01*

R-F18456-21CJ uCosminexus TP1/NET/TCP/IP 07-02*

R-F1B456-11CJ uCosminexus TP1/NET/TCP/IP(64) 07-02*

For Solaris 8, Solaris 9, and Solaris 10

P-9D64-3F31 uCosminexus TP1/NET/High Availability 07-00

P-9D64-8531 uCosminexus TP1/Extension 1 07-00

P-9D64-8931 uCosminexus TP1/High Availability 07-00

R-19451-216 uCosminexus TP1/Client/W 07-00

R-19452-216 uCosminexus TP1/Server Base 07-00

R-19453-216 uCosminexus TP1/FS/Direct Access 07-00

R-19454-216 uCosminexus TP1/FS/Table Access 07-00

R-19455-216 uCosminexus TP1/Message Control 07-03*

R-19456-216 uCosminexus TP1/NET/Library 07-04*

R-19459-216 uCosminexus TP1/Offline Tester 07-00

R-1945A-216 uCosminexus TP1/Online Tester 07-00

R-1945C-216 uCosminexus TP1/Shared Table Access 07-00

R-1945D-216 uCosminexus TP1/Resource Manager Monitor 07-00

R-1945E-216 uCosminexus TP1/Multi 07-00

R-F19456-2156 uCosminexus TP1/NET/XMAP3 07-01*

R-F19456-21C6 uCosminexus TP1/NET/TCP/IP 07-02*

For Red Hat Enterprise Linux AS 4 (AMD64 & Intel EM64T), Red Hat Enterprise Linux AS 4 (x86), Red Hat Enterprise Linux ES 4 (AMD64 & Intel EM64T), and Red Hat Enterprise Linux ES 4 (x86)

P-9S64-2161 uCosminexus TP1/Server Base 07-00

P-9S64-2351 uCosminexus TP1/FS/Direct Access 07-00

P-9S64-2451 uCosminexus TP1/FS/Table Access 07-00

P-9S64-2551 uCosminexus TP1/Client/W 07-00

P-9S64-3151 uCosminexus TP1/Message Control 07-00

P-9S64-3251 uCosminexus TP1/NET/Library 07-00

P-9S64-C371 uCosminexus TP1/Message Queue 07-01

P-F9S64-3251C uCosminexus TP1/NET/TCP/IP 07-00

P-F9S64-3251U uCosminexus TP1/NET/User Datagram Protocol 07-00

R-1845F-A15 uCosminexus TP1/Web 07-00

For Red Hat Enterprise Linux AS 4 (AMD64 & Intel EM64T), Red Hat Enterprise Linux AS 4 (x86), Red Hat Enterprise Linux ES 4 (AMD64 & Intel EM64T), Red Hat Enterprise Linux ES 4 (x86), Red Hat Enterprise Linux 5 (AMD/Intel 64), Red Hat Enterprise Linux 5 (x86), Red Hat Enterprise Linux 5 Advanced Platform (AMD/Intel 64), and Red Hat Enterprise Linux 5 Advanced Platform (x86)

P-9S64-2951 uCosminexus TP1/High Availability 07-00

P-9S64-8551 uCosminexus TP1/Extension 1 07-00

P-9S64-C771 uCosminexus TP1/Message Queue - Access 07-01

P-F9S64-3251D uCosminexus TP1/NET/High Availability 07-00

For Red Hat Enterprise Linux 5 (AMD/Intel 64), Red Hat Enterprise Linux 5 (x86), Red Hat Enterprise Linux 5 Advanced Platform (AMD/Intel 64), and Red Hat Enterprise Linux 5 Advanced Platform (x86)

P-9S64-2171 uCosminexus TP1/Server Base 07-03

P-9S64-2361 uCosminexus TP1/FS/Direct Access 07-03

P-9S64-2461 uCosminexus TP1/FS/Table Access 07-03

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

P-9S64-3161 uCosminexus TP1/Message Control 07-03*

P-9S64-3261 uCosminexus TP1/NET/Library 07-04*

P-9S64-C571 uCosminexus TP1/Message Queue 07-01

P-F9S64-32611 uCosminexus TP1/NET/User Agent 07-00

P-F9S64-3261C uCosminexus TP1/NET/TCP/IP 07-02

P-F9S64-3261U uCosminexus TP1/NET/User Datagram Protocol 07-00

For Red Hat Enterprise Linux 5 (AMD/Intel 64) and Red Hat Enterprise Linux 5 Advanced Platform (AMD/Intel 64)

P-9W64-2111 uCosminexus TP1/Server Base(64) 07-03

P-9W64-2311 uCosminexus TP1/FS/Direct Access(64) 07-03

P-9W64-2411 uCosminexus TP1/FS/Table Access(64) 07-03

P-9W64-2911 uCosminexus TP1/High Availability(64) 07-02

P-9W64-8511 uCosminexus TP1/Extension 1(64) 07-02

For Red Hat Enterprise Linux AS 4 (IPF)

P-9V64-2121 uCosminexus TP1/Server Base 07-00

P-9V64-2321 uCosminexus TP1/FS/Direct Access 07-00

P-9V64-2421 uCosminexus TP1/FS/Table Access 07-00

P-9V64-2521 uCosminexus TP1/Client/W 07-00

P-9V64-3121 uCosminexus TP1/Message Control 07-00

P-9V64-3221 uCosminexus TP1/NET/Library 07-00

P-9V64-C371 uCosminexus TP1/Message Queue(64) 07-01

P-9V64-C771 uCosminexus TP1/Message Queue - Access(64) 07-00

P-F9V64-3221C uCosminexus TP1/NET/TCP/IP 07-00

P-F9V64-3221U uCosminexus TP1/NET/User Datagram Protocol 07-00

For Red Hat Enterprise Linux AS 4 (IPF), Red Hat Enterprise Linux 5 (Intel Itanium), and Red Hat Enterprise Linux 5 Advanced Platform (Intel Itanium)

P-9V64-2921 uCosminexus TP1/High Availability 07-00

P-9V64-8521 uCosminexus TP1/Extension 1 07-00

P-F9V64-3221D uCosminexus TP1/NET/High Availability 07-00

For Red Hat Enterprise Linux 5 (Intel Itanium) and Red Hat Enterprise Linux 5 Advanced Platform (Intel Itanium)

P-9V64-2131 uCosminexus TP1/Server Base 07-02

P-9V64-2331 uCosminexus TP1/FS/Direct Access 07-02

P-9V64-2431 uCosminexus TP1/FS/Table Access 07-02

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

P-9V64-3131 uCosminexus TP1/Message Control 07-03*

P-9V64-3231 uCosminexus TP1/NET/Library 07-04*

P-F9V64-3231C uCosminexus TP1/NET/TCP/IP 07-02*

P-F9V64-3231U uCosminexus TP1/NET/User Datagram Protocol 07-00

For Windows 2000, Windows Server 2003, Windows Server 2003 x64 Editions, Windows Server 2003 R2, Windows Server 2003 R2 x64 Editions, Windows XP, Windows Vista, and Windows Vista x64

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

For Windows 2000, Windows Server 2003, Windows Server 2003 x64 Editions, Windows Server 2003 R2, Windows Server 2003 R2 x64 Editions, and Windows XP

R-1845F-8134 uCosminexus TP1/Web 07-00

For Windows 2000, Windows Server 2003, Windows Server 2003 x64 Editions, Windows Server 2003 R2, Windows Server 2003 R2 x64 Editions, Windows XP, Windows Vista, Windows Vista x64, Windows Server 2008, and Windows Server 2008 x64

P-2464-7824 uCosminexus TP1/Client for .NET Framework 07-03

R-15451-21 uCosminexus TP1/Connector for .NET Framework 07-03

For Windows Server 2003, Windows Server 2003 x64 Editions, Windows Server 2003 R2, Windows Server 2003 R2 x64 Editions, Windows XP, Windows Vista, Windows Vista x64, Windows Server 2008, and Windows Server 2008 x64

P-2464-2274 uCosminexus TP1/Server Base 07-03*

P-2464-2374 uCosminexus TP1/FS/Direct Access 07-03*

P-2464-2474 uCosminexus TP1/FS/Table Access 07-03*

P-2464-2544 uCosminexus TP1/Extension 1 07-00

P-2464-3154 uCosminexus TP1/Message Control 07-03*

P-2464-3254 uCosminexus TP1/NET/Library 07-04*

P-2464-3354 uCosminexus TP1/Messaging 07-00

P-2464-C374 uCosminexus TP1/Message Queue 07-01

P-2464-C774 uCosminexus TP1/Message Queue - Access 07-00

P-F2464-3254C uCosminexus TP1/NET/TCP/IP 07-02*

R-15452-21 uCosminexus TP1/Extension for .NET Framework 07-00

R-1945B-24 uCosminexus TP1/LiNK 07-02

For Windows Server 2003, Windows Server 2003 x64 Editions, Windows Server 2003 R2, Windows Server 2003 R2 x64 Editions, and Windows XP

P-F2464-32545 uCosminexus TP1/NET/XMAP3 07-01*

For Windows Server 2003, Windows Server 2003 x64 Editions, Windows Server 2003 R2, Windows Server 2003 R2 x64 Editions, Windows Server 2008, and Windows Server 2008 x64

P-2464-2934 uCosminexus TP1/High Availability 07-00

P-F2464-3254D uCosminexus TP1/NET/High Availability 07-00

For Java VM

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

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

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

These products were developed under a quality management system that has received ISO9001 and TickIT certification.

■ Trademarks

Printed in Japan.

■ Edition history

Edition 1 (3000-3-D54(E)): June 2006

Edition 3 (3000-3-D54-30(E)): October 2010

■ Copyright

Summary of amendments

The following table lists changes in this manual (3000-3-D54-30(E)) and product changes related to this manual for uCosminexus TP1/Server Base 07-03, uCosminexus TP1/Server Base(64) 07-03, uCosminexus TP1/Message Control 07-03, uCosminexus TP1/Message Control(64) 07-03, uCosminexus TP1/NET/Library 07-04, and uCosminexus TP1/NET/Library(64) 07-04.

Changes	Location
Explanations have been added about the maximum length of segments that can be sent or received.	*Message exchange processing (dc_mcf_~)* in *Chapter 2* *dc_mcf_execap*, *dc_mcf_receive*, *dc_mcf_recvsync*, *dc_mcf_reply*, *dc_mcf_send*, *dc_mcf_sendrecv*, *dc_mcf_sendsync*
Explanations have been added about global domains.	*Remote procedure call (dc_rpc_~)* in *Chapter 2* *dc_rpc_call_to*, *DCRPC_BINDTBL_SET*, *DCRPC_DIRECT_SCHEDULE*
Tables listing interface changes have been added to assist in migrating from TP1/Message Control Version 6 and earlier.	*Appendix B*

The following table lists changes in this manual (3000-3-D54-30(E)) and product changes related to this manual for uCosminexus TP1/Message Control 07-02 and uCosminexus TP1/NET/Library 07-03

Changes	Location
A library function can now be used to delete application timer startup requests. To support this change, the following function has been added: • `dc_mcf_adltap`	*1.1.1*, *1.1.1(2)*, *1.1.1(3)*, *Message exchange processing (dc_mcf_~)* in *Chapter 2* *dc_mcf_adltap*
Library functions can now be used to display the status of connections and to establish and release connections. To support this change, the following functions have been added: • `dc_mcf_tactcn` • `dc_mcf_tdctcn` • `dc_mcf_tlscn`	*1.1.1*, *1.1.1(2)*, *1.1.1(3)*, *Message exchange processing (dc_mcf_~)* in *Chapter 2* *dc_mcf_tactcn*, *dc_mcf_tdctcn*, *dc_mcf_tlscn*
A library function can now be used to display the status of MCF communication services and application startup services. To support this change, the following function has been added: • `dc_mcf_tlscom`	*1.1.1*, *1.1.1(2)*, *1.1.1(3)*, *Message exchange processing (dc_mcf_~)* in *Chapter 2* *dc_mcf_tlscom*

Changes	Location
Library functions can now be used to display the status of logical terminals, to shut down logical terminals, to release logical terminals from shutdown status, and to delete the output queue of logical terminals. To support this change, the following functions have been added: • dc_mcf_tactle • dc_mcf_tdctle • dc_mcf_tdlqle • dc_mcf_tlsle	*1.1.1*, *1.1.1(2)*, *1.1.1(3)*, *Message exchange processing (dc_mcf_~)* in *Chapter 2* *dc_mcf_tactle*, *dc_mcf_tdctle*, *dc_mcf_tdlqle*, *dc_mcf_tlsle*
A library function can now be used to acquire the acceptance status of connection establishment requests. To support this change, the following function has been added: • dc_mcf_tlsln	*1.1.1*, *1.1.1(2)*, *1.1.1(3)* *Message exchange processing (dc_mcf_~)* in *Chapter 2* *dc_mcf_tlsln*
Library functions can now be used to start and stop acceptance of server-type connection establishment requests. To support this change, the following functions have been added: • dc_mcf_tofln • dc_mcf_tonln	*1.1.1*, *1.1.1(2)*, *1.1.1(3)*, *Message exchange processing (dc_mcf_~)* in *Chapter 2* *dc_mcf_tofln*, *dc_mcf_tonln*
MHPs can now use the facility for dynamic loading of service functions.	*1.2.1(3)*, *1.2.5(2)(d)*

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

The following table lists changes in the manual (3000-3-D54-20(E)) and product changes related to that manual for uCosminexus TP1/Server Base 07-02, uCosminexus TP1/Message Control 07-01, and uCosminexus TP1/NET/Library 07-01.

Changes
An audit log output function was added. To support this change, the dc_log_audit_print function was added.
A facility that allows service functions to be loaded dynamically was added.
A function that allows the system to operate without the use of system journal files (journal fileless mode) was added. To support this change, some function arguments, return values, and notes were changed.
The description of the remote API facility was changed. To support this change, return values were changed or added.

The following table lists changes in the manual (3000-3-D54-20(E)) and product changes related to that manual for uCosminexus TP1/Server Base 07-01.

Changes
The C language interface in the 32-bit architecture and the C language interface in the 64-bit architecture have been unified.

Changes
Notes and return values have been added.

Preface

This manual explains the syntax of dedicated library functions which can be used with the OpenTP1 application programs. The program products of OpenTP1 are as follows:

- Distributed transaction processing facility TP1/Server Base
- Distributed application server TP1/LiNK

In this manual, an application program which is created by the user is abbreviated to a UAP (User Application Program).

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

Intended readers

This manual is intended for programmers who create user application programs (UAPs) used with TP1/Server Base or TP1/LiNK.

Readers of this manual are assumed to have knowledge about operating systems, online systems, handling of the machine to be used, and the syntax of the C language (ANSI C, C++, or Classic C) used for coding application programs.

This manual assumes that the reader has read the *OpenTP1 Programming Guide*.

Organization of this manual

This manual is organized as follows:

1. Creating Application Programs

Explains the procedure for creating application programs to be used with the OpenTP1.

2. Syntax of OpenTP1 Library Functions

Explains the syntax of the OpenTP1 library functions.

3. Syntax of OpenTP1 Library Functions (Message Log Reporting)

Explains the syntax of the OpenTP1 library functions for receiving message logs to obtain OpenTP1 statuses.

4. X/Open-compliant Application Programming Interface

Explains the syntax of the library functions complying with the X/Open.

5. Syntax of OpenTP1 Library Functions (Association Status Notification)

An SPP for a communication event is required for the client/server communication that uses the OSI TP communication protocol. This chapter explains the library functions used by SPPs for communication event and the formats of receive communication events.

6. X/Open-compliant Inter-application Communication (TxRPC)

Explains the syntax of Inter-Application communication (TxRPC) complying with the X/Open.

7. Coding Samples

Gives coding samples for OpenTP1 application programs.

8. Reference for Application Activation

Explains the communication facilities in the message exchange configuration, focusing on user exit routines relating to application program activate and MCF event (ERREVT4) reference information.

Appendix *A. Using OpenTP1 Remote Procedure Calls and XATMI-interfaced Functions in Combination*

Explains the procedures for creating UAPs that use OpenTP1 remote procedure calls and XATMI interface functions in combination.

Appendix *B. Changes to the Interfaces (for Migrating from Version 6 or Earlier)*

Provides tables that list changes in the interfaces to assist in migrating to Version 7 from Version 6 or earlier.

Related publications

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

OpenTP1 products

- *OpenTP1 Version 7 Description* (3000-3-D50(E))
- *OpenTP1 Version 7 Programming Guide* (3000-3-D51(E))
- *OpenTP1 Version 7 System Definition* (3000-3-D52(E))
- *OpenTP1 Version 7 Operation* (3000-3-D53(E))
- *OpenTP1 Version 7 Programming Reference C Language* (3000-3-D54(E))
- *OpenTP1 Version 7 Programming Reference COBOL Language* (3000-3-D55(E))
- *OpenTP1 Version 7 Messages* (3000-3-D56(E))

- *OpenTP1 Version 7 Tester and UAP Trace User's Guide* (3000-3-D57(E))
- *OpenTP1 Version 7 TP1/Client User's Guide TP1/Client/W, TP1/Client/P* (3000-3-D58(E))
- *OpenTP1 Version 7 TP1/Client User's Guide TP1/Client/J* (3000-3-D59(E))
- *OpenTP1 Version 7 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

- *Indexed Sequential Access Method ISAM* (3000-3-046(E))
- *XP/W* (3000-3-047(E))
- *Extended Mapping Service 2/Workstation XMAP2/W DESCRIPTION/USER'S GUIDE* (3000-7-421(E))
- *SEWB 3 General Information* (3000-7-450(E))
- *Job Management Partner 1/Base User's Guide* (3020-3-K06(E))
- *Job Management Partner 1/Base Messages* (3020-3-K07(E))
- *Job Management Partner 1/Base Software Developer's Guide* (3020-3-K08(E))

For OpenTP1 protocol manuals, please check whether English versions are available.

#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 for product names

This manual uses the following abbreviations for product names:

Abbreviation			Full name or meaning
AIX			AIX 5L V5.1
			AIX 5L V5.2
			AIX 5L V5.3
			AIX V6.1
Client .NET		TP1/Client for .NET Framework	uCosminexus TP1/Client for .NET Framework
Connector .NET		TP1/Connector for .NET Framework	uCosminexus TP1/Connector for .NET Framework
DPM			JP1/ServerConductor/Deployment Manager
HI-UX/WE2			HI-UX/workstation Extended Version 2
HP-UX	HP-UX (IPF)		HP-UX 11i V2 (IPF)
			HP-UX 11i V3 (IPF)
	HP-UX (PA-RISC)		HP-UX 11i V1 (PA-RISC)
			HP-UX 11i V2 (PA-RISC)
IPF			Itanium(R) Processor Family
Java			Java™
JP1	JP1/AJS2	JP1/AJS2 - Agent	JP1/Automatic Job Management System 2 - Agent
		JP1/AJS2 - Manager	JP1/Automatic Job Management System 2 - Manager
		JP1/AJS2 - View	JP1/Automatic Job Management System 2 - View
	JP1/AJS2 - Scenario Operation	JP1/AJS2 - Scenario Operation Manager	JP1/Automatic Job Management System 2 - Scenario Operation Manager
		JP1/AJS2 - Scenario Operation View	JP1/Automatic Job Management System 2 - Scenario Operation View
		JP1/NETM/Audit	JP1/NETM/Audit - Manager
Linux			Linux(R)
Linux (AMD64/Intel EM64T/x86)			Red Hat Enterprise Linux AS 4 (AMD64 & Intel EM64T)
			Red Hat Enterprise Linux AS 4 (x86)
			Red Hat Enterprise Linux ES 4 (AMD64 & Intel EM64T)

Abbreviation		Full name or meaning
		Red Hat Enterprise Linux ES 4 (x86)
		Red Hat Enterprise Linux 5 (AMD/Intel 64)
		Red Hat Enterprise Linux 5 (x86)
		Red Hat Enterprise Linux 5 Advanced Platform (AMD/Intel 64)
		Red Hat Enterprise Linux 5 Advanced Platform (x86)
Linux (IPF)		Red Hat Enterprise Linux AS 4 (IPF)
		Red Hat Enterprise Linux 5 (Intel Itanium)
		Red Hat Enterprise Linux 5 Advanced Platform (Intel Itanium)
MS-DOS		Microsoft(R) MS-DOS(R)
NETM/DM		JP1/NETM/DM Client
		JP1/NETM/DM Manager
		JP1/NETM/DM SubManager
Oracle		Oracle 10g
		Oracle9i
Solaris		Solaris 8
		Solaris 9
		Solaris 10
TP1/Client	TP1/Client/J	uCosminexus TP1/Client/J
	TP1/Client/P	uCosminexus TP1/Client/P
	TP1/Client/W	uCosminexus TP1/Client/W
		uCosminexus TP1/Client/W(64)
TP1/EE		uCosminexus TP1/Server Base Enterprise Option
		uCosminexus TP1/Server Base Enterprise Option(64)
TP1/Extension 1		uCosminexus TP1/Extension 1
		uCosminexus TP1/Extension 1(64)

Abbreviation		Full name or meaning
TP1/FS/Direct Access		uCosminexus TP1/FS/Direct Access
		uCosminexus TP1/FS/Direct Access(64)
TP1/FS/Table Access		uCosminexus TP1/FS/Table Access
		uCosminexus TP1/FS/Table Access(64)
TP1/High Availability		uCosminexus TP1/High Availability
		uCosminexus TP1/High Availability(64)
TP1/LiNK		uCosminexus TP1/LiNK
TP1/Message Control		uCosminexus TP1/Message Control
		uCosminexus TP1/Message Control(64)
TP1/Message Control/Tester		uCosminexus TP1/Message Control/Tester
TP1/Message Queue		uCosminexus TP1/Message Queue
		uCosminexus TP1/Message Queue(64)
TP1/Message Queue - Access		uCosminexus TP1/Message Queue - Access
		uCosminexus TP1/Message Queue - Access(64)
TP1/Messaging		uCosminexus TP1/Messaging
TP1/Multi		uCosminexus TP1/Multi
TP1/NET/HDLC		uCosminexus TP1/NET/HDLC
TP1/NET/High Availability		uCosminexus TP1/NET/High Availability
		uCosminexus TP1/NET/High Availability(64)
TP1/NET/HSC		uCosminexus TP1/NET/HSC
TP1/NET/Library		uCosminexus TP1/NET/Library
		uCosminexus TP1/NET/Library(64)
TP1/NET/NCSB		uCosminexus TP1/NET/NCSB
TP1/NET/OSAS-NIF		uCosminexus TP1/NET/OSAS-NIF
TP1/NET/OSI-TP		uCosminexus TP1/NET/OSI-TP
TP1/NET/SLU - TypeP2	TP1/NET/ Secondary Logical Unit - TypeP2	uCosminexus TP1/NET/Secondary Logical Unit - TypeP2

Abbreviation	Full name or meaning
TP1/NET/TCP/IP	uCosminexus TP1/NET/TCP/IP
	uCosminexus TP1/NET/TCP/IP(64)
TP1/NET/UDP	uCosminexus TP1/NET/User Datagram Protocol
TP1/NET/User Agent	uCosminexus TP1/NET/User Agent
TP1/NET/X25	uCosminexus TP1/NET/X25
TP1/NET/X25-Extended	uCosminexus TP1/NET/X25-Extended
TP1/NET/XMAP3	uCosminexus TP1/NET/XMAP3
TP1/Offline Tester	uCosminexus TP1/Offline Tester
TP1/Online Tester	uCosminexus TP1/Online Tester
TP1/Resource Manager Monitor	uCosminexus TP1/Resource Manager Monitor
TP1/Server Base	uCosminexus TP1/Server Base
	uCosminexus TP1/Server Base(64)
TP1/Shared Table Access	uCosminexus TP1/Shared Table Access
TP1/Web	uCosminexus TP1/Web
Windows 2000	Microsoft(R) Windows(R) 2000 Advanced Server Operating System
	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
Windows Server 2003	Microsoft(R) Windows Server(R) 2003, Datacenter Edition
	Microsoft(R) Windows Server(R) 2003, Enterprise Edition
	Microsoft(R) Windows Server(R) 2003, Standard Edition
Windows Server 2003 R2	Microsoft(R) Windows Server(R) 2003 R2, Enterprise Edition
	Microsoft(R) Windows Server(R) 2003 R2, Standard Edition
Windows Server 2003 x64 Editions	Microsoft(R) Windows Server(R) 2003, Datacenter x64 Edition

Abbreviation	Full name or meaning
	Microsoft(R) Windows Server(R) 2003, Enterprise x64 Edition
	Microsoft(R) Windows Server(R) 2003, Standard x64 Edition
Windows Server 2003 R2 x64 Editions	Microsoft(R) Windows Server(R) 2003 R2, Enterprise x64 Edition
	Microsoft(R) Windows Server(R) 2003 R2, Standard x64 Edition
Windows Server 2008	Microsoft(R) Windows Server(R) 2008 Datacenter (x86)
	Microsoft(R) Windows Server(R) 2008 Enterprise (x86)
	Microsoft(R) Windows Server(R) 2008 Standard (x86)
Windows Server 2008 x64 Editions	Microsoft(R) Windows Server(R) 2008 Datacenter (x64)
	Microsoft(R) Windows Server(R) 2008 Enterprise (x64)
	Microsoft(R) Windows Server(R) 2008 Standard (x64)
Windows Vista	Microsoft(R) Windows Vista(R) Business (x86)
	Microsoft(R) Windows Vista(R) Enterprise (x86)
	Microsoft(R) Windows Vista(R) Ultimate (x86)
Windows Vista x64 Editions	Microsoft(R) Windows Vista(R) Business (x64)
	Microsoft(R) Windows Vista(R) Enterprise (x64)
	Microsoft(R) Windows Vista(R) Ultimate (x64)
Windows XP	Microsoft(R) Windows(R) XP Professional Operating System

- The term Windows is used to indicate Windows Server 2003, Windows XP and Windows Vista if the difference in functions among them need not be considered.
- The term UNIX is used to indicate AIX, HP-UX, Linux, and Solaris.

Conventions: Acronyms

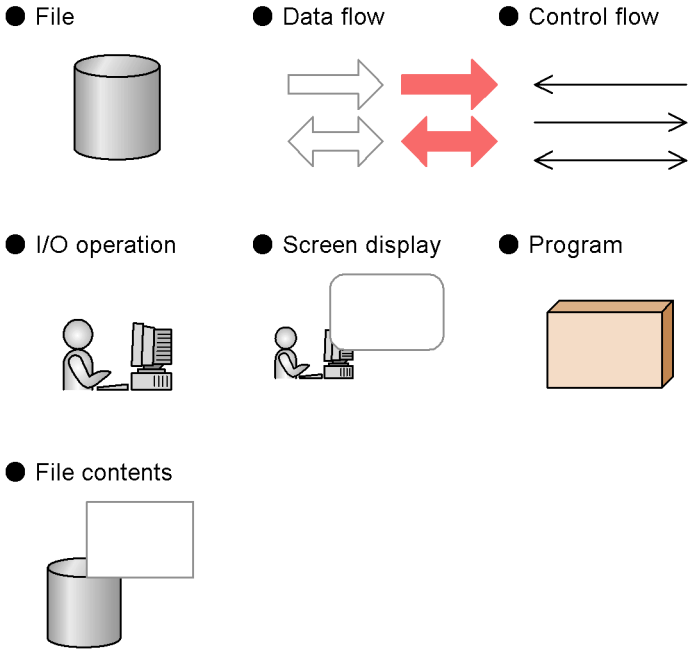
This manual also uses the following acronyms:

Acronym	Full name or meaning
ACL	Access Control List
ANSI	American National Standards Institute
AP	Application Program
API	Application Programming Interface
C/S	Client/Server
CRM	Communication Resource Manager
CUP	Client User Program
DAM	Direct Access Method
DBMS	Database Management System
DML	Data Manipulation Language
DNS	Domain Name System
FEP	Front End Processor
GUI	Graphical User Interface
HA	High Availability
ISAM	Indexed Sequential Access Method
IST	Internode Shared Table
LAN	Local Area Network
MCF	Message Control Facility
MHP	Message Handling Program
MQA	Message Queue Access
MQI	Message Queue Interface
OS	Operating System
OSI	Open Systems Interconnection
OSI TP	Open Systems Interconnection Transaction Processing
PC	Personal Computer
PRF	Performance
RM	Resource Manager

Acronym	Full name or meaning
RPC	Remote Procedure Call
SPP	Service Providing Program
SUP	Service Using Program
TAM	Table Access Method
TCP/IP	Transmission Control Protocol/Internet Protocol
UAP	User Application Program
UOC	User Own Coding
VM	Virtual Machine
WAN	Wide Area Network
WS	Workstation

Conventions: Diagrams

This manual uses the following conventions in diagrams:



Conventions: Differences between JIS and ASCII keyboards

The JIS code and ASCII code keyboards are different in the input characters represented by the following codes. In this manual, the use of a JIS keyboard is assumed for these characters.

Code	JIS keyboard	ASCII keyboard
$(5c)_{16}$	¥ (yen symbol)	\ (backslash)
$(7e)_{16}$	‾ (overline)	~ (tilde)

Conventions: Fonts and symbols

The following table explains the fonts used in this manual:

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: • From the **File** menu, choose **Open**. • Click the **Cancel** button. • In the **Enter name** entry box, type your name.
Italics	*Italics* are used to indicate a placeholder for some actual text to be provided by the user or system. For example: • Write the command as follows: `copy` *source-file target-file* • The following message appears: `A file was not found. (file =` *file-name*`)` *Italics* are also used for emphasis. For example: • 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.`

The following table explains the symbols used in this manual:

Symbol	Convention
|	In syntax explanations, a vertical bar separates multiple items, and has the meaning of OR. For example: `A|B|C` means A, or B, or C.

Symbol	Convention
{ }	In syntax explanations, curly brackets indicate that only one of the enclosed items is to be selected. For example: {A|B|C} means only one of A, or B, or C.
[ ]	In syntax explanations, square brackets indicate that the enclosed item or items are optional. For example: [A] means that you can specify A or nothing. [B|C] means that you can specify B, or C, or nothing.
. . .	In coding, an ellipsis (...) indicates that one or more lines of coding are not shown for purposes of brevity. In syntax explanations, an ellipsis indicates that the immediately preceding item can be repeated as many times as necessary. For example: A, B, B, ... means that, after you specify A, B, you can specify B as many times as necessary.
~	The item before this symbol must be specified according to the rule given in < > after this symbol.
< >	Information between these symbols is the syntax of the item.

Conventions for permitted characters

In most cases, only the following characters are permitted as syntax elements (if other characters are permitted, the manual will state this explicitly):

Type	Definition
Upper-case alphabetic characters	A to Z
Lower-case alphabetic characters	a to z
Alphabetic characters	A to Z, a to z
Numeric characters	0 to 9
Alphanumeric characters	A to Z, a to z, 0 to 9
Symbols	!, #, $, %, &, ', (,), *, +, -, ., /, :, ;, <, =, >, ?, @, [, \,], ^, _, `, {, |, }, and ~
Path name	String that is composed of alphanumeric characters, slashes (/), and periods (.) and conforms to the rule under the OS used.
Service group name	Must be an ASCII character string of up to 31 bytes. Note that null characters, spaces, at marks (@), and periods cannot be used. When a service group name is specified in a data area, it must end with a space. This space will not be included in the length of the character string.

Type	Definition
Service name	Must be an ASCII character string of up to 31 bytes. Note that null and space characters cannot be used. When a service name is specified in a data area, it must end with a space. This space will not be included in the length of the character string.
Physical file name	Must be a path name consisting of the special file name followed by a name of 14 or less bytes. The entire path name must not exceed 63 characters.
Logical file name	Must be an alphanumeric character string of 1 to 8 bytes that begins with an alphabetic character.

Conventions: KB, MB, GB, and TB

This manual uses the following conventions:

- 1 KB (kilobyte) is 1,024 bytes.
- 1 MB (megabyte) is $1{,}024^2$ bytes.
- 1 GB (gigabyte) is $1{,}024^3$ bytes.
- 1 TB (terabyte) is $1{,}024^4$ bytes.

Conventions: Platform-specific notational differences

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

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

Conventions: Version numbers

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*.

Acknowledgments

This manual contains information from the X/Open CAE Specification System Interfaces and Headers, Issue 4, (C202 ISBN 1-872630-47-2) Copyright (C) July 1992, X/Open Company Limited pursuant to approval from this company.

Part of that information is based on IEEE Std 1003.1-1990, (C) 1990 Institute of Electrical and Electronics Engineers, Inc. and IEEE Std 1003.2/D12, (C) 1992 Institute of Electrical and Electronics Engineers, Inc.

Any part of this manual that is copyrighted by the companies above may not be copied or reproduced in any form without prior approval gained from the copyright holders.

This manual contains information from the X/Open Preliminary Specification Distributed Transaction Processing: The TxRPC Specification, (P305 ISBN 1-85912-000-8) Copyright (C) July 1993, X/Open Company Limited, pursuant to approval from X/Open Company Limited.

Any part of this manual that is copyrighted by the above mentioned company may not be copied or reproduced in any form without prior approval gained from the copyright holder.

This manual contains information that is copyrighted by Open Software Foundation, Inc.

This document and the software described herein are furnished under a license, and may be used and copied only in accordance with the terms of such license and with the inclusion of the above copyright notice. Title to and ownership of the document and software remain with OSF or its licensors.

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

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

- Chapter *4. X/Open-Compliant Application Programming Interface*

4.1 XATMI-Interfaced Application Programming Interface (tp~)

Quotations from X/Open CAE Specification Distributed Transaction Processing: The TX (Transaction Demarcation) Specification published by X/Open Company Limited

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

- Chapter *4. X/Open-Compliant Application Programming Interface*

4.2 TX-Interfaced Application Programming Interface (tx_~)

Quotations from X/Open Preliminary Specification Distributed Transaction Processing: The TxRPC Specification published by X/Open Company Limited

The following chapter comes from parts of Chapter *1. Introduction*, *Chapter 2 Model and Definitions*, and *Chapter 3. Interface Overview* of the above document.

- *Chapter 6. X/Open-Compliant Inter-Application Communication (TxRPC)*

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

List of figures

List of tables

Chapter

1. Creating Application Programs

This chapter outlines how to create OpenTP1 application programs in the C language.

This chapter contains the following sections:

1.1 Coding application program

1.1.1 Relationship between application programs and functions

The table below shows the correspondences between the OpenTP1 library functions and their facilities.

Table 1-1: Functions in OpenTP1 library and their facilities

Facility classification	OpenTP1 function names and facilities	
System operation management	dc_adm_call_command	Execute an operation command.
	dc_adm_complete	Report the completion of user server start processing.
	dc_adm_status	Report the status of a user server.
Multinode facility	dc_adm_get_nd_status	Acquire the status of a specified OpenTP1 node.
	dc_adm_get_nd_status_begin	Start acquiring the status of an OpenTP1 node.
	dc_adm_get_nd_status_done	Terminate acquiring the status of an OpenTP1 node.
	dc_adm_get_nd_status_next	Acquire the status of an OpenTP1 node.
	dc_adm_get_nodeconf_begin	Start acquiring a node identifier.
	dc_adm_get_nodeconf_done	Terminate acquiring a node identifier.
	dc_adm_get_nodeconf_next	Acquire a node identifier.
	dc_adm_get_node_id	Acquire the node identifier of the local node.
	dc_adm_get_sv_status	Acquire the status of a specified user server.
	dc_adm_get_sv_status_begin	Start acquiring the status of a user server.
	dc_adm_get_sv_status_done	Terminate acquiring the status of a user server.
	dc_adm_get_sv_status_next	Acquire the status of a user server.
DAM file service	dc_dam_bseek	Seek a physical file block.

Facility classification	OpenTP1 function names and facilities	
	dc_dam_close	Close a logical file.
	dc_dam_create	Allocate a physical file.
	dc_dam_dget	Input directly a physical file block.
	dc_dam_dput	Output directly a physical file block.
	dc_dam_end	Terminate using an unrecoverable DAM file.
	dc_dam_get	Input a physical file block.
	dc_dam_hold	Shut down a logical file.
	dc_dam_iclose	Close a physical file.
	dc_dam_iopen	Open a physical file.
	dc_dam_open	Open a logical file.
	dc_dam_put	Output a physical file block.
	dc_dam_read	Input a logical file block.
	dc_dam_release	Release a logical file from the shutdown state.
	dc_dam_rewrite	Update a logical file block.
	dc_dam_start	Start using an unrecoverable DAM file.
	dc_dam_status	Reference the status of a logical file.
	dc_dam_write	Output a logical file block.
IST service	dc_ist_close	Close an internode shared table.
	dc_ist_open	Open an internode shared table.
	dc_ist_read	Input an internode shared table record.
	dc_ist_write	Output an internode shared table record.
User journal acquisition	dc_jnl_ujput	Acquire a user journal.
Lock for resources	dc_lck_get	Enable locking of a resource.
	dc_lck_release_all	Release all the resources from lock.

Facility classification	OpenTP1 function names and facilities	
	dc_lck_release_byname	Release resource from lock specified by name.
Audit log output	dc_log_audit_print	Output audit log data.
Message log output	dc_logprint	Output message log.
Message exchange processing	dc_mcf_adltap	Delete an application timer start request.
	dc_mcf_ap_info	Report the application information.
	dc_mcf_ap_info_uoc	Report application information to a user exit routine.
	dc_mcf_close	Close the MCF environment.
	dc_mcf_commit	Commit an MHP.
	dc_mcf_contend	Terminate continuous-inquiry response processing.
	dc_mcf_execap	Activate an application program.
	dc_mcf_mainloop	Start an MHP service.
	dc_mcf_open	Open the MCF environment.
	dc_mcf_receive	Receive a message.
	dc_mcf_recvsync	Receive a synchronous message.
	dc_mcf_reply	Send a response message.
	dc_mcf_resend	Resend a message.
	dc_mcf_rollback	Enable MHP rollback.
	dc_mcf_send	Send a message.
	dc_mcf_sendrecv	Exchange a synchronous message.
	dc_mcf_sendsync	Send a synchronous message.
	dc_mcf_tactcn	Establish a connection.
	dc_mcf_tactle	Release a logical terminal from shutdown status.
	dc_mcf_tdctcn	Release connection.
	dc_mcf_tdctle	Shut down a logical terminal.

Facility classification	OpenTP1 function names and facilities	
	dc_mcf_tdlqle	Delete a logical terminal's output queue.
	dc_mcf_tempget	Accept temporary-stored data.
	dc_mcf_tempput	Update temporary-stored data.
	dc_mcf_timer_set	Set user timer monitoring.
	dc_mcf_timer_cancel	Cancel user timer monitoring.
	dc_mcf_tlscn	Acquire a connection status.
	dc_mcf_tlscom	Acquire the status of MCF communication services.
	dc_mcf_tlsle	Acquire a logical terminal status.
	dc_mcf_tlsln	Acquire the acceptance status for a server-type connection establishment request.
	dc_mcf_tofln	Stop accepting server-type connection establishment requests.
	dc_mcf_tonln	Start accepting server-type connection establishment requests.
Performance verification trace	dc_prf_get_trace_num	Report the sequential number for an acquired performance verification trace.
	dc_prf_utrace_put	Acquire user-specific performance verification traces.
Remote API facility	dc_rap_connect	Establish a connection with a RAP-processing listener.
	dc_rap_disconnect	Release a connection with a RAP-processing listener.
Remote procedure call	dc_rpc_call	Request a remote service.
	dc_rpc_call_to	Invoke a remote service with a communication destination specified.
	dc_rpc_close	Terminate an application program.
	dc_rpc_cltsend	Report data to CUP unidirectionally.
	dc_rpc_discard_further_replies	Reject the receiving of processing results.

Facility classification	OpenTP1 function names and facilities	
	dc_rpc_discard_specific_reply	Reject acceptance of particular processing results.
	dc_rpc_get_callers_address	Acquire the node address of a client UAP.
	dc_rpc_get_error_descriptor	Acquire the descriptor of an asynchronous response-type RPC request which has encountered an error.
	dc_rpc_get_gateway_address	Acquire the node address of a gateway.
	dc_rpc_get_service_prio	Reference the schedule priority of a service request.
	dc_rpc_get_watch_time	Reference the service response waiting interval.
	dc_rpc_mainloop	Start an SPP service.
	dc_rpc_open	Start an application program.
	dc_rpc_poll_any_replies	Receive processing results in asynchronous mode.
	dc_rpc_service_retry	Retry a service function.
	dc_rpc_set_service_prio	Set a schedule priority of a service request.
	dc_rpc_set_watch_time	Update a service response waiting interval.
Real-time statistical information service	dc_rts_utrace_put	Acquire real-time statistical information for arbitrary section.
TAM file service	dc_tam_close	Close a TAM table.
	dc_tam_delete	Delete a TAM table record.
	dc_tam_get_inf	Acquire TAM table status.
	dc_tam_open	Open a TAM table.
	dc_tam_read	Input a TAM table record.
	dc_tam_read_cancel	Cancel the input of a TAM table record.

Facility classification	OpenTP1 function names and facilities	
	dc_tam_rewrite	Update a TAM table record on the assumption of input.
	dc_tam_status	Acquire TAM table information.
	dc_tam_write	Update/add a TAM table record.
Transaction control	dc_trn_begin	Start a transaction.
	dc_trn_chained_commit	Enable commitment in chained mode.
	dc_trn_chained_rollback	Enable rollback in chained mode.
	dc_trn_info	Report the information about the current transaction.
	dc_trn_unchained_commit	Enable commitment in unchained mode.
	dc_trn_unchained_rollback	Enable rollback in unchained mode.
Online tester management	dc_uto_test_status	Report the test status of a user server.

(1) Facilities and functions available with SUPs

The table below lists the facilities and functions which can be used with SUPs.

Table 1-2: Facilities and functions available with SUPs

Facility available with SUP		OpenTP1 function	SUP operating conditions	
			Outside the transaction processing range	Inside the transaction processing range
System operation management	Execute an operation command.	dc_adm_call_command	Y	Y
	Report the completion of user server start processing.	dc_adm_complete	Y	N
	Report the status of a user server.	dc_adm_status	Y	Y
Multinode facility	Acquire the status of a specified OpenTP1 node.	dc_adm_get_nd_status	Y	Y
	Start acquiring the status of an OpenTP1 node.	dc_adm_get_nd_status_begin	Y	Y

Facility available with SUP		OpenTP1 function	SUP operating conditions	
			Outside the transaction processing range	Inside the transaction processing range
	Terminate acquiring the status of an OpenTP1 node.	dc_adm_get_nd_status_done	Y	Y
	Acquire the status of an OpenTP1 node.	dc_adm_get_nd_status_next	Y	Y
	Start acquiring a node identifier.	dc_adm_get_nodeconf_begin	Y	Y
	Terminate acquiring a node identifier.	dc_adm_get_nodeconf_done	Y	Y
	Acquire a node identifier.	dc_adm_get_nodeconf_next	Y	Y
	Acquire the node identifier of the local node.	dc_adm_get_node_id	Y	Y
	Acquire the status of a specified user server.	dc_adm_get_sv_status	Y	Y
	Start acquiring the status of a user server.	dc_adm_get_sv_status_begin	Y	Y
	Terminate acquiring the status of a user server.	dc_adm_get_sv_status_done	Y	Y
	Acquire the status of a user server.	dc_adm_get_sv_status_next	Y	Y
DAM file service	Close a logical file.	dc_dam_close	Y	Y
	Terminate using an unrecoverable DAM file.	dc_dam_end	Y	Y
	Shut down a logical file.	dc_dam_hold	N	Y
	Open a logical file.	dc_dam_open	Y	Y
	Input a logical file block.	dc_dam_read	Y	Y
	Release a logical file from the shutdown state.	dc_dam_release	N	Y

Facility available with SUP		OpenTP1 function	SUP operating conditions	
			Outside the transaction processing range	Inside the transaction processing range
	Update a logical file block.	dc_dam_rewrite	(Y)	Y
	Start using an unrecoverable DAM file.	dc_dam_start	Y	Y
	Reference the status of a logical file.	dc_dam_status	Y	Y
	Output a logical file block.	dc_dam_write	(Y)	Y
IST service	Close an internode shared table.	dc_ist_close	Y	Y
	Open an internode shared table.	dc_ist_open	Y	Y
	Input an internode shared table record.	dc_ist_read	Y	Y
	Output an internode shared table record.	dc_ist_write	Y	Y
User journal acquisition	Acquire a user journal.	dc_jnl_ujput	Y	Y
Lock for resources	Enable locking of a resource.	dc_lck_get	N	Y
	Release all the resources from lock.	dc_lck_release_all	N	Y
	Release resource from lock specified by name.	dc_lck_release_byname	N	Y
Audit log output	Output audit log data.	dc_log_audit_print	Y	Y
Message log output	Output message log	dc_logprint	Y	Y
Performance verification trace	Report the sequential number for an acquired performance verification trace.	dc_prf_get_trace_num	Y	Y

Facility available with SUP		OpenTP1 function	SUP operating conditions	
			Outside the transaction processing range	Inside the transaction processing range
	Acquire user-specific performance verification traces.	dc_prf_utrace_put	Y	Y
Remote API facility	Establish a connection with a RAP-processing listener.	dc_rap_connect	Y	N
	Release a connection with a RAP-processing listener.	dc_rap_disconnect	Y	N
Remote procedure call	Request a remote service.	dc_rpc_call	Y	Y
	Invoke a remote service with a communication destination specified.	dc_rpc_call_to	Y	Y
	Terminate an application program.	dc_rpc_close	Y	N
	Reject the receiving of processing results.	dc_rpc_discard_further_replies	Y	Y
	Reject acceptance of particular processing results.	dc_rpc_discard_specific_reply	Y	Y
	Acquire the descriptor of an asynchronous response-type RPC request which has encountered an error.	dc_rpc_get_error_descriptor	Y	Y
	Reference the schedule priority of a service request.	dc_rpc_get_service_prio	Y	Y
	Reference the service response waiting interval.	dc_rpc_get_watch_time	Y	Y
	Start an application program.	dc_rpc_open	Y	N

Facility available with SUP		OpenTP1 function	SUP operating conditions	
			Outside the transaction processing range	Inside the transaction processing range
	Receive processing results in asynchronous mode.	dc_rpc_poll_any_replies	Y	Y
	Set a schedule priority of a service request.	dc_rpc_set_service_prio	Y	Y
	Change the response waiting interval of a service request.	dc_rpc_set_watch_time	Y	Y
Real-time statistical information service	Acquire real-time statistical information for arbitrary section.	dc_rts_utrace_put	Y	Y
TAM file service	Close a TAM table.	dc_tam_close	Y	Y
	Delete a TAM table record.	dc_tam_delete	N	Y
	Acquire TAM table status.	dc_tam_get_inf	Y	Y
	Open a TAM table.	dc_tam_open	Y	Y
	Input a TAM table record.	dc_tam_read	N	Y
	Cancel the input of a TAM table record.	dc_tam_read_cancel	N	Y
	Update a TAM table record on the assumption of input.	dc_tam_rewrite	N	Y
	Acquire TAM table information.	dc_tam_status	Y	Y
	Update/add a TAM table record.	dc_tam_write	N	Y
Transaction control	Start a transaction.	dc_trn_begin	Y	N
	Enable commitment in chained mode.	dc_trn_chained_commit	N	Y

Facility available with SUP		OpenTP1 function	SUP operating conditions	
			Outside the transaction processing range	Inside the transaction processing range
	Enable rollback in chained mode.	dc_trn_chained_rollback	N	Y
	Report the information about the current transaction.	dc_trn_info	Y	Y
	Enable commitment in unchained mode.	dc_trn_unchained_commit	N	Y
	Enable rollback in unchained mode.	dc_trn_unchained_rollback	N	Y
Online tester management	Report the test status of a user server.	dc_uto_test_status	Y	Y

Legend:

Y: Can be used with SUPs.

(Y): Can be used only in access to an unrecoverable DMA file.

N: Cannot be used with SUPs.

(2) Facilities and functions available with SPPs

The table below lists the facilities and functions which can be used with SPPs.

Table 1-3: Facilities and functions available with SPPs

Facility available with SPP		OpenTP1 function	SPP operating conditions		
			Outside the transaction processing range	Inside the transaction processing range	
				Root	Not root
System operation management	Execute an operation command.	dc_adm_call_command	Y	Y	Y
	Report the status of a user server.	dc_adm_status	Y	Y	Y

Facility available with SPP		OpenTP1 function	SPP operating conditions		
			Outside the transaction processing range	Inside the transaction processing range	
				Root	Not root
Multinode facility	Acquire the status of a specified OpenTP1 node.	dc_adm_get_nd_status	Y	Y	Y
	Start acquiring the status of an OpenTP1 node.	dc_adm_get_nd_status_begin	Y	Y	Y
	Terminate acquiring the status of an OpenTP1 node.	dc_adm_get_nd_status_done	Y	Y	Y
	Acquire the status of an OpenTP1 node.	dc_adm_get_nd_status_next	Y	Y	Y
	Start acquiring a node identifier.	dc_adm_get_nodeconf_begin	Y	Y	Y
	Terminate acquiring a node identifier.	dc_adm_get_nodeconf_done	Y	Y	Y
	Acquire a node identifier.	dc_adm_get_nodeconf_next	Y	Y	Y
	Acquire the node identifier of the local node.	dc_adm_get_node_id	Y	Y	Y
	Acquire the status of a specified user server.	dc_adm_get_sv_status	Y	Y	Y
	Start acquiring the status of a user server.	dc_adm_get_sv_status_begin	Y	Y	Y
	Terminate acquiring the status of a user server.	dc_adm_get_sv_status_done	Y	Y	Y
	Acquire the status of a user server.	dc_adm_get_sv_status_next	Y	Y	Y
DAM file service	Close a logical file.	dc_dam_close	Y	Y	Y
	Terminate using an unrecoverable DAM file.	dc_dam_end	Y	Y	Y

Facility available with SPP		OpenTP1 function	SPP operating conditions		
			Outside the transaction processing range	Inside the transaction processing range	
				Root	Not root
	Shut down a logical file.	dc_dam_hold	N	Y	Y
	Open a logical file.	dc_dam_open	Y	Y	Y
	Input a logical file block.	dc_dam_read	N	Y	Y
	Release a logical file from the shutdown state.	dc_dam_release	N	Y	Y
	Update a logical fileblock.	dc_dam_rewrite	(Y)	Y	Y
	Start using an unrecoverable DAM file.	dc_dam_start	Y	Y	Y
	Reference the status of a logical file.	dc_dam_status	Y	Y	Y
	Output a logical file block.	dc_dam_write	(Y)	Y	Y
IST service	Close an internode shared table.	dc_ist_close	Y	Y	Y
	Open an internode shared table.	dc_ist_open	Y	Y	Y
	Input an internode shared table record.	dc_ist_read	Y	Y	Y
	Output an internode shared table record.	dc_ist_write	Y	Y	Y
User journal acquisition	Acquire a user journal.	dc_jnl_ujput	Y	Y	Y
Lock for resources	Enable locking of a resource.	dc_lck_get	N	Y	Y
	Release all the resources from lock.	dc_lck_release_all	N	Y	Y
	Release resource from lock specified by name.	dc_lck_release_byname	N	Y	Y

Facility available with SPP		OpenTP1 function	SPP operating conditions		
			Outside the transaction processing range	Inside the transaction processing range	
				Root	Not root
Audit log output	Output audit log data.	dc_log_audit_print	Y	Y	Y
Message log output	Output message log.	dc_logprint	Y	Y	Y
Message exchange processing	Delete an application timer start request.	dc_mcf_adltap	Y	Y	Y
	Close the MCF environment.	dc_mcf_close	O	N	N
	Activate an application program.	dc_mcf_execap	N	Y	Y
	Open the MCF environment.	dc_mcf_open	O	N	N
	Receive a synchronous message.	dc_mcf_recvsync	Y	Y	Y
	Resend a message.	dc_mcf_resend	N	Y	Y
	Send a message.	dc_mcf_send	N	Y	Y
	Exchange a synchronous message.	dc_mcf_sendrecv	Y	Y	Y
	Send a synchronous message.	dc_mcf_sendsync	Y	Y	Y
	Establish a connection.	dc_mcf_tactcn	Y	Y	Y
	Release a logical terminal from shutdown status.	dc_mcf_tactle	Y	Y	Y
	Release connection.	dc_mcf_tdctcn	Y	Y	Y
	Shut down a logical terminal.	dc_mcf_tdctle	Y	Y	Y
	Delete a logical terminal's output queue.	dc_mcf_tdlqle	Y	Y	Y

Facility available with SPP		OpenTP1 function	SPP operating conditions		
			Outside the transaction processing range	Inside the transaction processing range	
				Root	Not root
	Set user timer monitoring.	dc_mcf_timer_set	Y	Y	Y
	Cancel user timer monitoring.	dc_mcf_timer_cancel	Y	Y	Y
	Acquire a connection status.	dc_mcf_tlscn	Y	Y	Y
	Acquire the status of MCF communication services.	dc_mcf_tlscom	Y	Y	Y
	Acquire a logical terminal status.	dc_mcf_tlsle	Y	Y	Y
	Acquire the acceptance status for a server-type connection establishment request.	dc_mcf_tlsln	Y	Y	Y
	Stop accepting server-type connection establishment requests.	dc_mcf_tofln	Y	Y	Y
	Start accepting server-type connection establishment requests.	dc_mcf_tonln	Y	Y	Y
Performance verification trace	Report the sequential number for an acquired performance verification trace.	dc_prf_get_trace_num	Y	Y	Y
	Acquire user-specific performance verification traces.	dc_prf_utrace_put	Y	Y	Y
Remote API facility	Establish a connection with a RAP-processing listener.	dc_rap_connect	Y	N	N

Facility available with SPP		OpenTP1 function	SPP operating conditions		
			Outside the transaction processing range	Inside the transaction processing range	
				Root	Not root
	Release a connection with a RAP-processing listener.	dc_rap_disconnect	Y	N	N
Remote procedure call	Request a remote service.	dc_rpc_call	Y	Y	Y
	Invoke a remote service with a communication destination specified.	dc_rpc_call_to	Y	Y	Y
	Terminate an application program.	dc_rpc_close	O	N	N
	Report data to CUP unidirectionally.	dc_rpc_cltsend	Y	Y	Y
	Reject the receiving of processing results.	dc_rpc_discard_further_replies	Y	Y	Y
	Reject acceptance of particular processing results.	dc_rpc_discard_specific_reply	Y	Y	Y
	Acquire the node address of a client UAP.	dc_rpc_get_callers_address	Y	Y	Y
	Acquire the descriptor of an asynchronous response-type RPC request which has encountered an error.	dc_rpc_get_error_descriptor	Y	Y	Y
	Acquire the node address of a gateway.	dc_rpc_get_gateway_address	Y	Y	Y
	Reference the schedule priority of a service request.	dc_rpc_get_service_prio	Y	Y	Y
	Reference the service response waiting interval.	dc_rpc_get_watch_time	Y	Y	Y
	Start an SPP service.	dc_rpc_mainloop	O	N	N

Facility available with SPP		OpenTP1 function	SPP operating conditions		
			Outside the transaction processing range	Inside the transaction processing range	
				Root	Not root
	Start an application program.	dc_rpc_open	O	N	N
	Receive processing results in asynchronous mode.	dc_rpc_poll_any_replies	Y	Y	Y
	Retry a service function.	dc_rpc_service_retry	Y	N	N
	Set a schedule priority of a service request.	dc_rpc_set_service_prio	Y	Y	Y
	Update the response waiting interval of a service request.	dc_rpc_set_watch_time	Y	Y	Y
Real-time statistical information service	Acquire real-time statistical information for arbitrary section.	dc_rts_utrace_put	Y	Y	Y
TAM file service	Close a TAM table.	dc_tam_close	Y	Y	Y
	Delete a TAM table record.	dc_tam_delete	N	Y	Y
	Acquire TAM table status.	dc_tam_get_inf	Y	Y	Y
	Open a TAM table.	dc_tam_open	Y	Y	Y
	Input a TAM table record.	dc_tam_read	N	Y	Y
	Cancel the input of a TAM table record.	dc_tam_read_cancel	N	Y	Y
	Update a TAM table record on the assumption of input.	dc_tam_rewrite	N	Y	Y
	Acquire TAM table information.	dc_tam_status	Y	Y	Y

Facility available with SPP		OpenTP1 function	SPP operating conditions		
			Outside the transaction processing range	Inside the transaction processing range	
				Root	Not root
	Update/add a TAM table record.	dc_tam_write	N	Y	Y
Transaction control	Start a transaction.	dc_trn_begin	Y	N	N
	Enable commitment in chained mode.	dc_trn_chained_commit	N	Y	N
	Enable rollback in chained mode.	dc_trn_chained_rollback	N	Y	N
	Report the information about the current transaction.	dc_trn_info	Y	Y	Y
	Enable commitment in unchained mode.	dc_trn_unchained_commit	N	Y	N
	Enable rollback in unchained mode.	dc_trn_unchained_rollback	N	Y	Y
Online tester management	Report the test status of a user server.	dc_uto_test_status	Y	Y	Y

Legend:

Y: Can be used with SPPs.

(Y): Can be used only in access to an unrecoverable DAM file.

N: Cannot be used with SPPs.

O: Can be used only from the main function.

Note

Root means the root transaction branch, and *Not root* means a transaction branch other than the root transaction branch.

(3) Facilities and functions available with MHPs

The table below lists the facilities and functions which can be used with MHPs.

Table 1-4: Facilities and functions available with MHPs

Facility available with MHP		OpenTP1 function	MHP operating conditions	
			Outside the transaction processing range	Inside the transaction processing range
System operation management	Execute an operation command.	dc_adm_call_command	Y	Y
	Report the status of a user server.	dc_adm_status	Y	Y
Multinode facility	Acquire the status of a specified OpenTP1 node.	dc_adm_get_nd_status	Y	Y
	Start acquiring the status of an OpenTP1 node.	dc_adm_get_nd_status_begin	Y	Y
	Terminate acquiring the status of an OpenTP1 node.	dc_adm_get_nd_status_done	Y	Y
	Acquire the status of an OpenTP1 node.	dc_adm_get_nd_status_next	Y	Y
	Start acquiring a node identifier.	dc_adm_get_nodeconf_begin	Y	Y
	Terminate acquiring a node identifier.	dc_adm_get_nodeconf_done	Y	Y
	Acquire a node identifier.	dc_adm_get_nodeconf_next	Y	Y
	Acquire the node identifier of the local node.	dc_adm_get_node_id	Y	Y

Facility available with MHP		OpenTP1 function	MHP operating conditions	
			Outside the transaction processing range	Inside the transaction processing range
	Acquire the status of a specified user server.	dc_adm_get_sv_status	Y	Y
	Start acquiring the status of a user server.	dc_adm_get_sv_status_begin	Y	Y
	Terminate acquiring the status of a user server.	dc_adm_get_sv_status_done	Y	Y
	Acquire the status of a user server.	dc_adm_get_sv_status_next	Y	Y
DAM file service	Close a logical file.	dc_dam_close	Y	Y
	Terminate using an unrecoverable DAM file.	dc_dam_end	Y	Y
	Shut down a logical file.	dc_dam_hold	N	Y
	Open a logical file.	dc_dam_open	Y	Y
	Input a logical file block.	dc_dam_read	Y	Y
	Release a logical file from the shutdown state.	dc_dam_release	N	Y
	Update a logical file block.	dc_dam_rewrite	(Y)	Y

Facility available with MHP		OpenTP1 function	MHP operating conditions	
			Outside the transaction processing range	Inside the transaction processing range
	Start using an unrecoverable DAM file.	dc_dam_start	Y	Y
	Reference the status of a logical file.	dc_dam_status	Y	Y
	Output a logical file block.	dc_dam_write	(Y)	Y
IST service	Close an internode shared table.	dc_ist_close	Y	Y
	Open an internode shared table.	dc_ist_open	Y	Y
	Input an internode shared table record.	dc_ist_read	Y	Y
	Output an internode shared table record.	dc_ist_write	Y	Y
User journal acquisition	Acquire a user journal.	dc_jnl_ujput	Y	Y
Lock for resources	Enable locking of a resource.	dc_lck_get	N	Y
	Release all the resources from lock.	dc_lck_release_all	N	Y
	Release resource from lock specified by name.	dc_lck_release_by_name	N	Y

Facility available with MHP		OpenTP1 function	MHP operating conditions	
			Outside the transaction processing range	Inside the transaction processing range
Audit log output	Output audit log data.	dc_log_audit_print	Y	Y
Message log output	Output message log.	dc_logprint	Y	Y
Message exchange processing	Delete an application timer start request.	dc_mcf_adltap	Y	Y
	Report the application information.	dc_mcf_ap_info	NO	Y
	Close the MCF environment.	dc_mcf_close	O	O
	Commit an MHP.	dc_mcf_commit	N	Y
	Terminate continuous-inquiry response processing.	dc_mcf_contend	NO	Y
	Activate an application program.	dc_mcf_execap	NO	Y
	Start an MHP service.	dc_mcf_mainloop	O	N
	Open the MCF environment.	dc_mcf_open	O	O
	Receive a message.	dc_mcf_receive	NO	Y
	Receive a synchronous message.	dc_mcf_recvsync	Y	Y

Facility available with MHP		OpenTP1 function	MHP operating conditions	
			Outside the transaction processing range	Inside the transaction processing range
	Send a response message.	dc_mcf_reply	NO	Y
	Resend a message	dc_mcf_resend	N	Y
	Enable MHP rollback.	dc_mcf_rollback	N	Y
	Send a message.	dc_mcf_send	NO	Y
	Exchange a synchronous message.	dc_mcf_sendrecv	Y	Y
	Send a synchronous message.	dc_mcf_sendsync	Y	Y
	Establish a connection.	dc_mcf_tactcn	Y	Y
	Release a logical terminal from shutdown status.	dc_mcf_tactle	Y	Y
	Release connection	dc_mcf_tdctcn	Y	Y
	Shut down a logical terminal.	dc_mcf_tdctle	Y	Y
	Delete the output queue of a logical terminal.	dc_mcf_tdlqle	Y	Y
	Accept temporary-stored data.	dc_mcf_tempget	NO	Y
	Update temporary-stored data.	dc_mcf_tempput	NO	Y

Facility available with MHP		OpenTP1 function	MHP operating conditions	
			Outside the transaction processing range	Inside the transaction processing range
	Set user timer monitoring.	dc_mcf_timer_set	Y	Y
	Cancel user timer monitoring.	dc_mcf_timer_cancel	Y	Y
	Acquire the connection status.	dc_mcf_tlscn	Y	Y
	Acquire the MCF communication service status.	dc_mcf_tlscom	Y	Y
	Acquire the logical terminal status.	dc_mcf_tlsle	Y	Y
	Acquire the acceptance status for a server-type connection establishment request.	dc_mcf_tlsln	Y	Y
	Stop accepting server-type connection establishment requests.	dc_mcf_tofln	Y	Y
	Start accepting server-type connection establishment requests.	dc_mcf_tonln	Y	Y

Facility available with MHP		OpenTP1 function	MHP operating conditions	
			Outside the transaction processing range	Inside the transaction processing range
Performance verification trace	Report the sequential number for an acquired performance verification trace.	dc_prf_get_trace_num	Y	Y
	Acquire user-specific performance verification traces.	dc_prf_utrace_put	Y	Y
Remote API facility	Establish a connection with a RAP-processing listener.	dc_rap_connect	Y	N
	Release a connection with a RAP-processing listener.	dc_rap_disconnect	Y	N
Remote procedure call	Request a remote service	dc_rpc_call	Y	Y
	Invoke a remote service with a communication destination specified.	dc_rpc_call_to	Y	Y
	Terminate an application program.	dc_rpc_close	O	N
	Report data to CUP unidirectionally.	dc_rpc_cltsend	Y	Y

Facility available with MHP		OpenTP1 function	MHP operating conditions	
			Outside the transaction processing range	Inside the transaction processing range
	Reject the receiving of processing results.	dc_rpc_discard_further_replies	Y	Y
	Reject acceptance of particular processing results.	dc_rpc_discard_specific_reply	Y	Y
	Acquire the descriptor of an asynchronous response-type RPC request which has encountered an error.	dc_rpc_get_error_descriptor	Y	Y
	Reference the schedule priority of a service request.	dc_rpc_get_service_prio	Y	Y
	Reference the service response waiting interval.	dc_rpc_get_watch_time	Y	Y
	Start an application program.	dc_rpc_open	O	N
	Receive processing results in asynchronous mode.	dc_rpc_poll_any_replies	Y	Y
	Set a schedule priority of a service request.	dc_rpc_set_service_prio	Y	Y

Facility available with MHP		OpenTP1 function	MHP operating conditions	
			Outside the transaction processing range	Inside the transaction processing range
	Update the response waiting interval of a service request.	dc_rpc_set_watch_time	Y	Y
Real-time statistical information service	Acquire real-time statistical information for arbitrary section.	dc_rts_utrace_put	Y	Y
TAM file service	Close a TAM table.	dc_tam_close	Y	Y
	Delete a TAM table record.	dc_tam_delete	N	Y
	Acquire TAM table status.	dc_tam_get_inf	Y	Y
	Open a TAM table.	dc_tam_open	Y	Y
	Input a TAM table record.	dc_tam_read	N	Y
	Cancel the input of a TAM table record.	dc_tam_read_cancel	N	Y
	Update a TAM table record on the assumption of input.	dc_tam_rewrite	N	Y
	Acquire TAM table information.	dc_tam_status	Y	Y
	Update/add a TAM table record.	dc_tam_write	N	Y

Facility available with MHP		OpenTP1 function	MHP operating conditions	
			Outside the transaction processing range	Inside the transaction processing range
Transaction control	Start a transaction.	dc_trn_begin	O	N
	Report the information about the current transaction.	dc_trn_info	Y	Y
	Enable commitment in unchained mode.	dc_trn_unchained_commit	N	O
	Enable rollback in unchained mode.	dc_trn_unchained_rollback	N	O
Online tester management	Report the test status of a user server.	dc_uto_test_status	Y	Y

Legend:

Y: Can be used with MHPs.

(Y): Can be used only in access to an unrecoverable DAM file.

O: Can be used only from the main function.

NO: The function can be used only in the service-function range of nontransaction attribute MHPs.

N: Cannot be used with MHPs.

Note

"Outside the transaction processing range" means the range of nontransaction attribute MHPs or MHP main functions.

(4) Facilities and functions available with UAPs that handles offline work

The table below lists the facilities and functions which can be used with UAPs that handles offline work.

Table 1-5: Facilities and functions available with UAPs that handles offline work

Facility available with UAP that handles offline work		OpenTP1 function
DAM file service	Seek a physical file block.	dc_dam_bseek
	Allocate a physical file.	dc_dam_create
	Input directly a physical file block.	dc_dam_dget
	Output directly a physical file block.	dc_dam_dput
	Input a physical file block.	dc_dam_get
	Close a physical file.	dc_dam_iclose
	Open a physical file.	dc_dam_iopen
	Output a physical file block.	dc_dam_put
Performance verification trace	Report the sequential number for an acquired performance verification trace.	dc_prf_get_trace_num
	Acquire user-specific performance verification traces.	dc_prf_utrace_put

1.1.2 Coding rules

(1) Notes on coding

For OpenTP1, a UAP can be created in either C or C++ language. If you are using C language, code the UAP according to the ANSI C format or the pre-ANSI K&R format. If you are using C++ language, code the UAP in conformance with the C++ language specifications. Although the availability of some functions in the provided standard library is limited, most functions in the library can be used together with the functions in the OpenTP1 library.

In addition, any system calls and program libraries can also be used. However, it is recommendable to use OS-provided standard functions and system calls when writing UAPs in order to assure high portability of the UAPs.

When creating UAPs which use system calls and arbitrary program libraries, note the following:

1. When using a signal from the UAP, do not register the type of a signal handler (SIGILL or SIGBUS) which creates a core file during operation with the signal default specified. If the signal handler is registered, a core file is not created even when the program terminates abnormally. As a result, troubleshooting is impossible.
2. When using a signal from the UAP, do not use a function in the OpenTP1 library

from the signal handler.

3. Do not use the following system call:
   - `chdir` (change of the current working directory)
4. Do not use the following system calls after the function `dc_rpc_open()`:
   - `fork` (new process creation)
   - `exec` (file execution)
   - `system` (shell command issuance)
5. Do not use jump functions (`setjmp` and `longjmp`) which extend over functions in the C-language library.
6. When using another program library, do not use Xlib and OSF/Motif functions which control event-driven dispatching.

If the OS is HP-UX, always specify `immediate` as the bind mode at linkage. If an executable file created as a bind mode other than `immediate` is used as an OpenTP1 UAP, the system operation is undefined. Use the OS chatr command to check whether the bind mode for the created UAP is `immediate`.

(2) Notes on naming

We recommend that you include a certain prefix character string in the names of any variables or definitions coded by the user. If any names duplicate those used by the OS or OpenTP1, system operation is unpredictable.

(a) Service function names

Service functions must be given names which are 20 or less alphanumeric characters in length and begin with an alphabetic character. Do not give service functions the following names:

- Names beginning with `dc`
- Names beginning with `CBLDC`
- Names beginning with `tx` or `TX`
- Names beginning with `tp` or `TP`

(b) External variable names

Do not give external variables the following names except when such names are used according to the instructions in this manual:

- Names beginning with `dc`
- Names beginning with `CBLDC`
- Names beginning with `tx` or `TX`

- Names beginning with `tp` or `TP`

(c) Constant names

Do not give the following names as constant names defined in `#define` statements except when such names are used according to the instructions in this manual:

- Names beginning with `DC`
- Names beginning with `CBLDC`
- Names beginning with `TX`
- Names beginning with `TP`

(3) Termination method

If the COBOL85 program has been executed even only once in a process of a UAP created in C language, use the `cblend` function to enable `exit`. If the UAP is terminated without using the `cblend` function, some information will not be output (such as the COBOL85 count information). See the corresponding COBOL language manual for details on the `cblend` function.

(4) When using Windows

Conform to the specifications of the C compiler used by Windows for compiling and linking UAPs when the OpenTP1 (TP1/LiNK) is used by Windows.

(5) When using TP1/Message Control

The source files of C user application programs and user exit routines used in Version 6 can also be used as is in Version 7 in the following cases: (1) when both Versions 6 and 7 are for the 32-bit architecture, and (2) when both Versions 6 and 7 are for the 64-bit architecture.

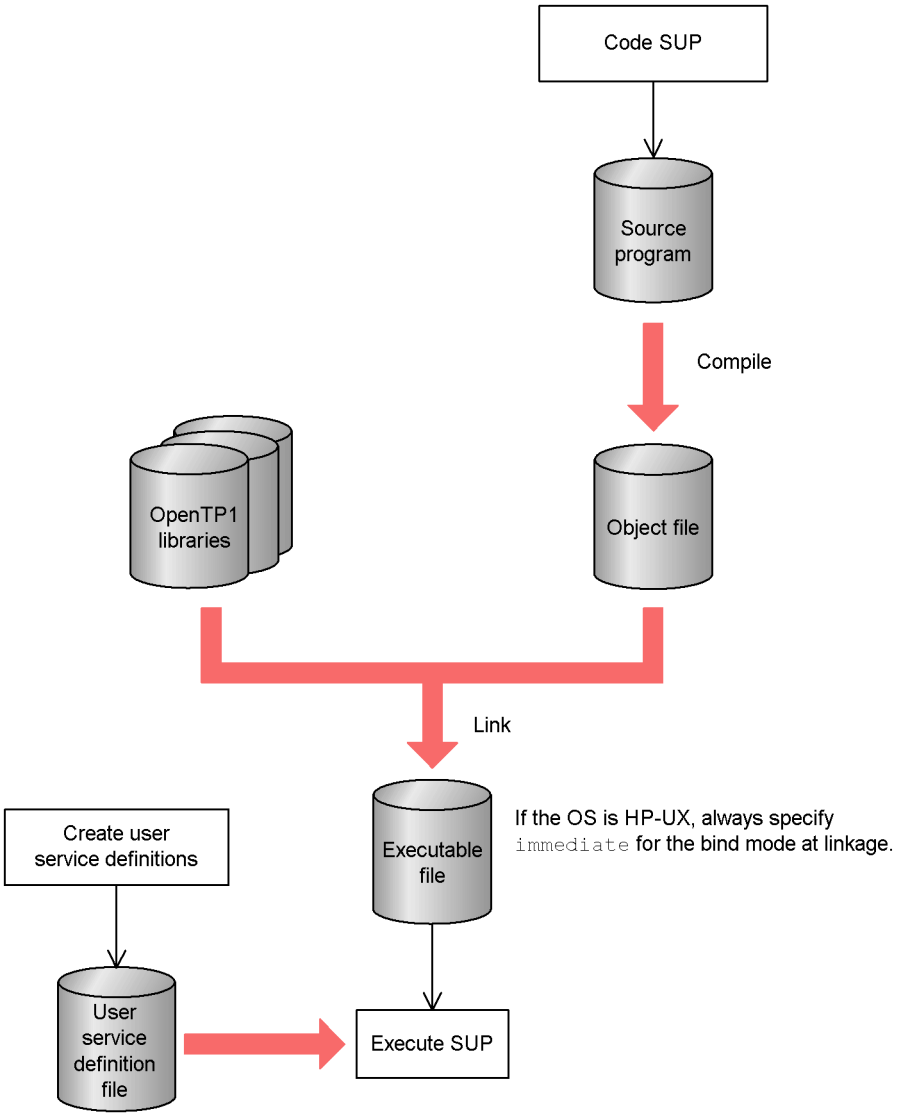
1.2 Creating application programs (TCP/IP)

1.2.1 Procedure for creating application programs

(1) General procedure for creating an SUP

The figure below shows the procedure for creating an SUP.

Figure 1-1: General procedure for creating SUPs



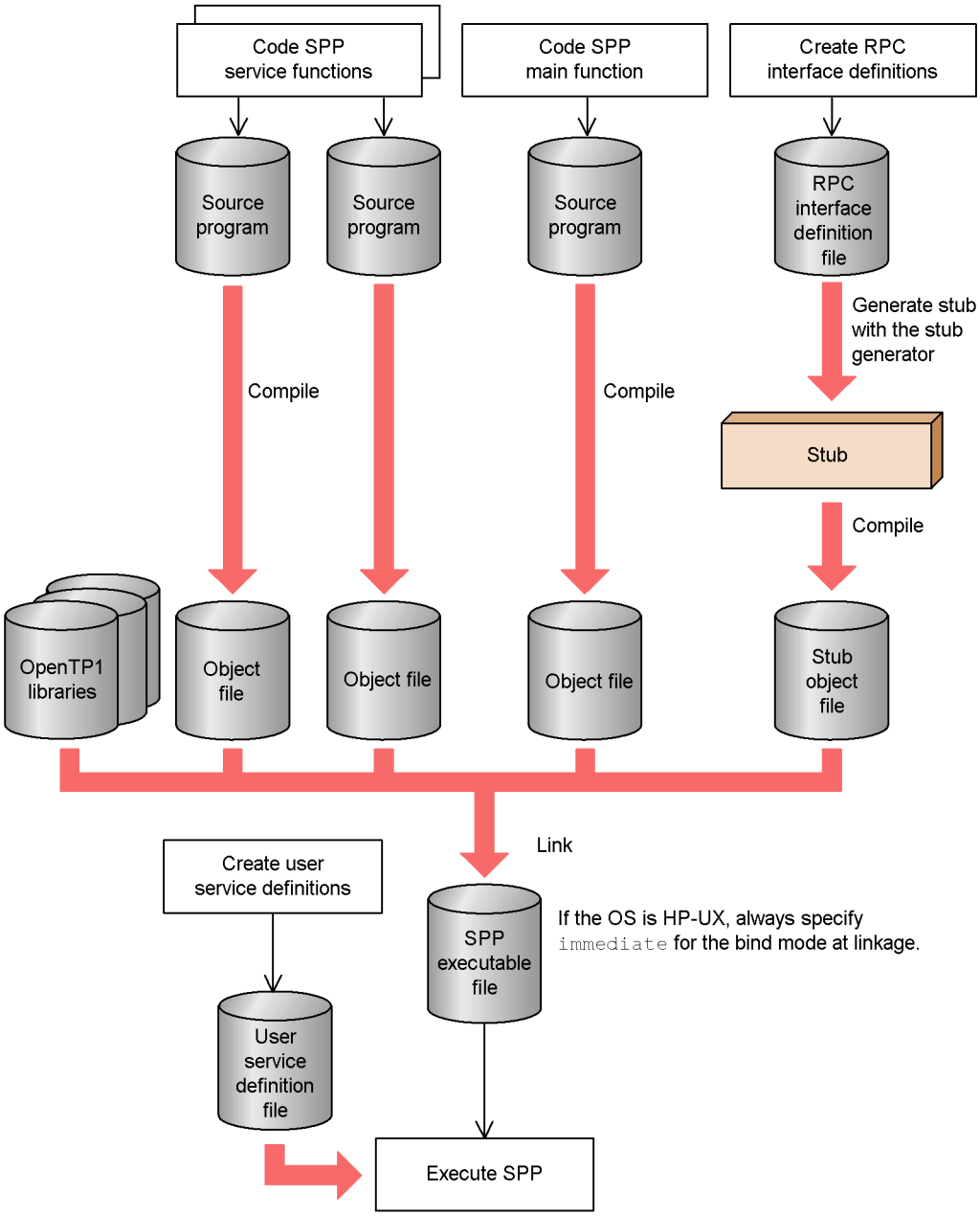
(2) General procedures for creating an SPP

The SPP creation procedure depends on whether the SPP uses a stub or uses dynamic loading of service functions.

(a) General procedure for creating an SPP (when using a stub)

The figure below shows the general procedure for creating an SPP by using a stub.

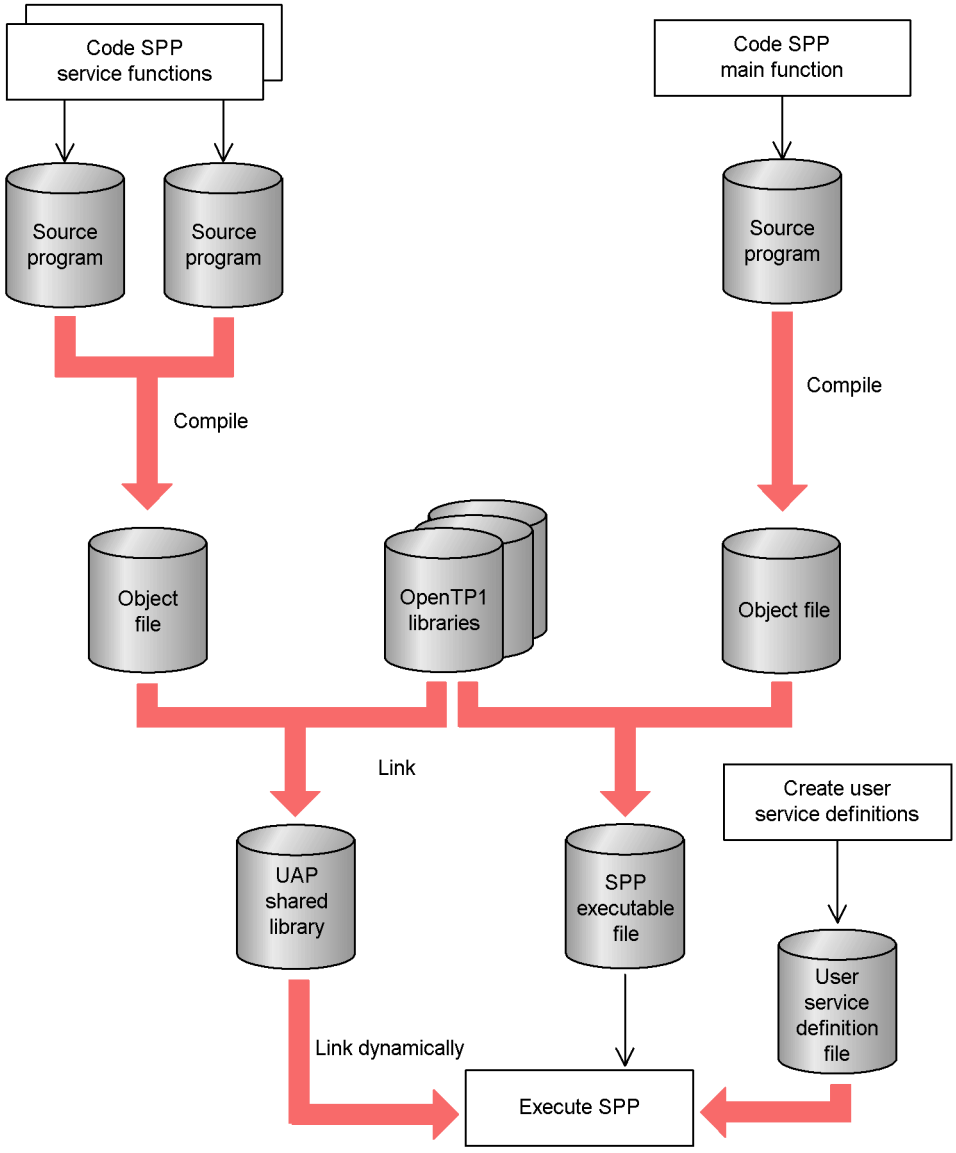
Figure 1-2: General procedure for creating an SPP (when using a stub)



(b) General procedure for creating an SPP (when using dynamic loading of service functions)

The following shows the general procedure for creating an SPP that dynamically loads service functions.

Figure 1-3: General procedure for creating an SPP (when using dynamic loading of service functions)



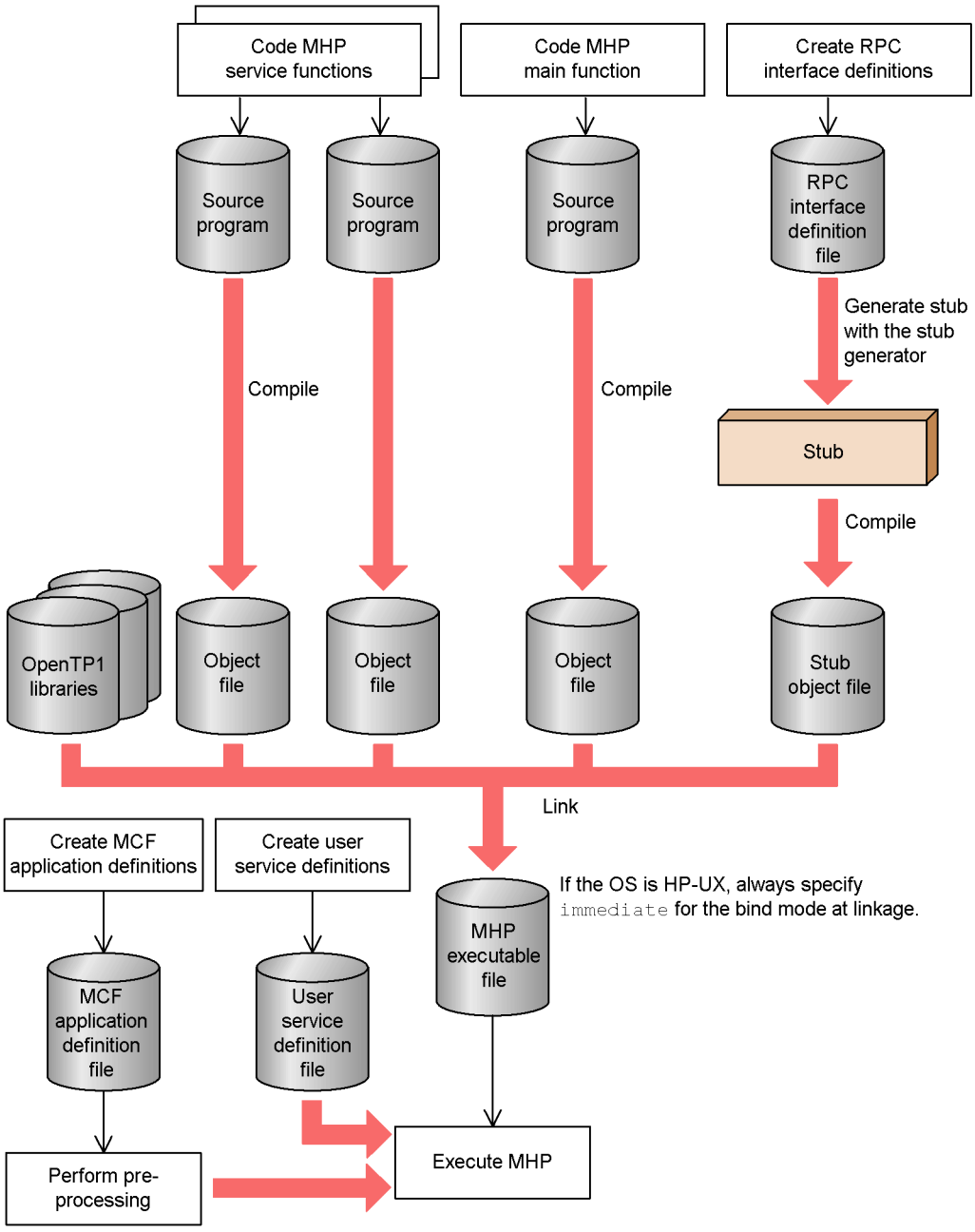
(3) General procedures for creating an MHP

The MHP creation procedure depends on whether the MHP uses a stub or uses dynamic loading of service functions.

(a) General procedure for creating an MHP (when using a stub)

The figure below shows the general procedure for creating an MHP that uses a stub.

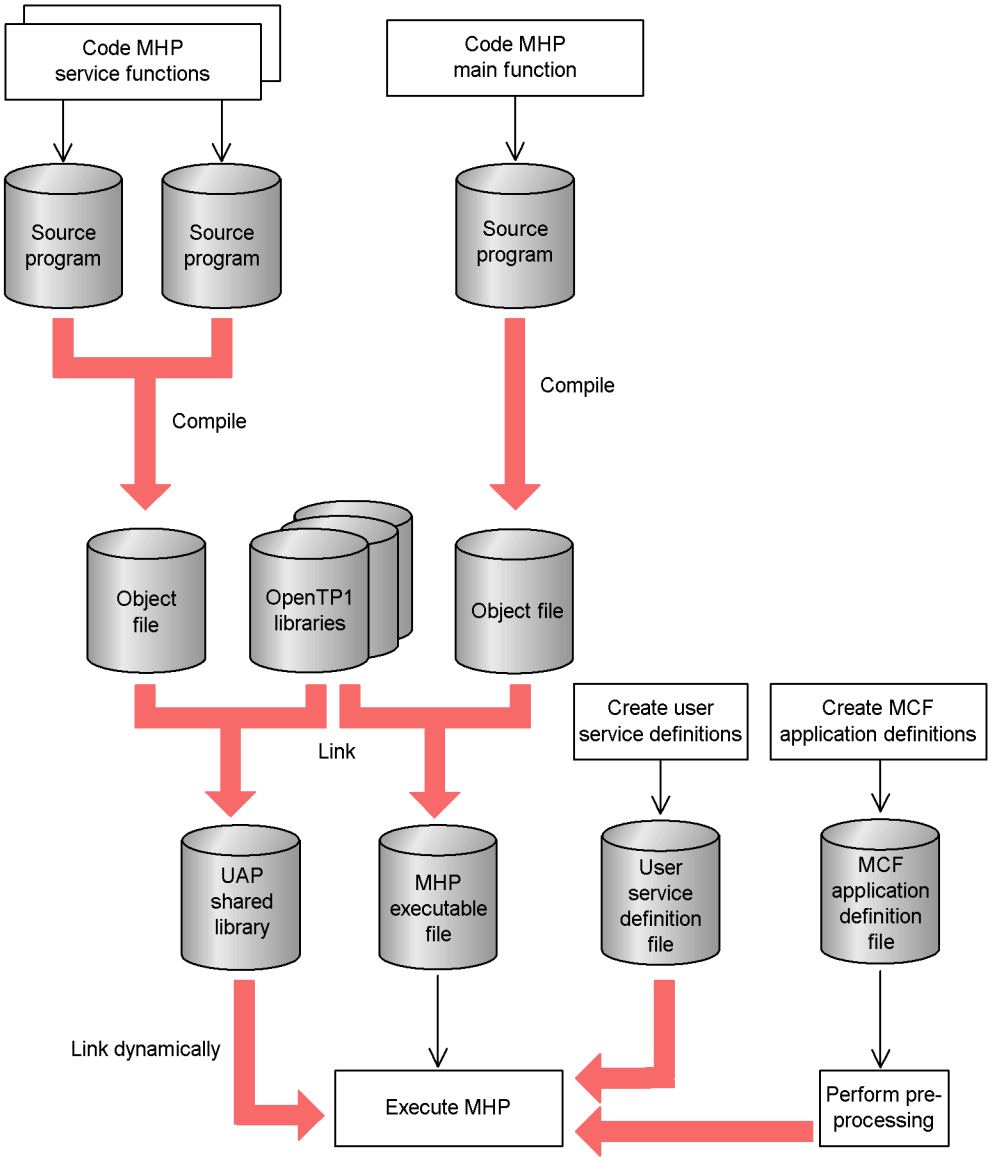
Figure 1-4: General procedure for creating an MHP (when using a stub)



(b) General procedure for creating an MHP (when using dynamic loading of service functions

The figure below shows the general procedure for creating an MHP that uses dynamic loading of service functions.

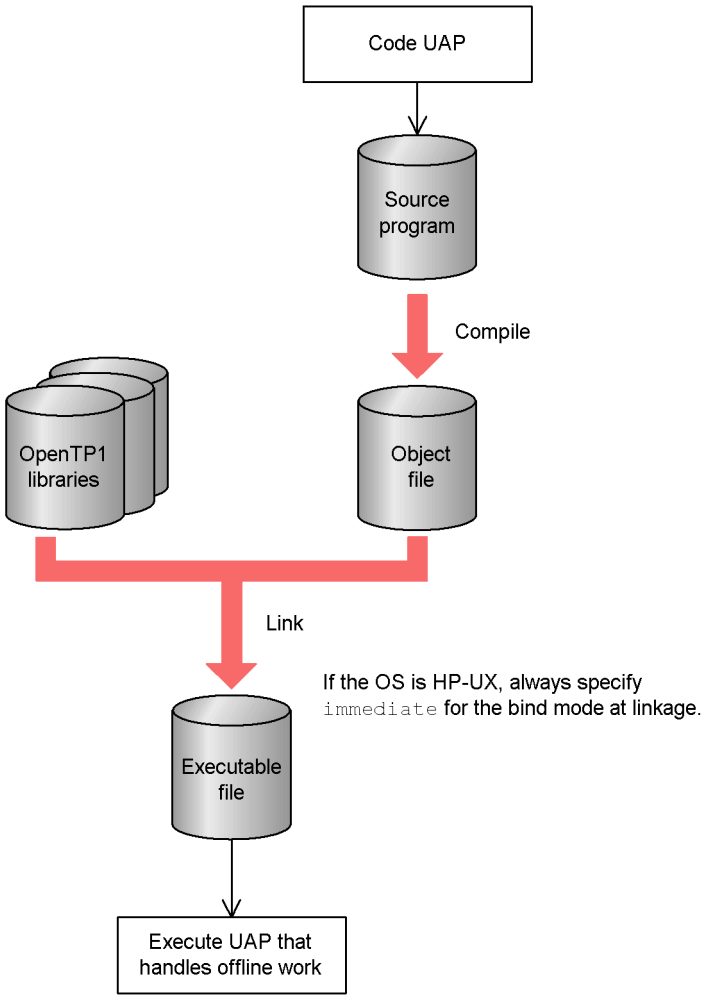
Figure 1-5: General procedure for creating an MHP (when using dynamic loading of service functions)



(4) General procedure for creating UAP that handles offline work

The figure below shows the general procedure for creating a UAP that handles offline work.

Figure 1-6: General procedure for creating a UAP that handles offline work



1.2.2 Creating stubs

UAPs used with the OpenTP1 require libraries for fulfilling inter-UAP service requests. One of these libraries is called a *stub*.

The explanation below deals with stubs of UAPs (SUP and SPP) which use an OpenTP1 remote procedure calls (`dc_rpc_call()`) and MHP stubs. See *1.3*

Creating XATMI interface application programs (TCP/IP, OSI TP) on how to create stubs which will be used when the XATMI interface is used for communication.

(1) Application programs requiring stubs

Among the UAPs used with the OpenTP1, UAPs having service functions (SPP and MHP) usually require a stub. However, a stub is not required if all service functions are put in the UAP shared library from which they are loaded dynamically. The UAP shared library is created by linking the UAP object files compiled from UAP source files.

Note that UAPs that handle offline work and SUPs do not require a stub because they do not have a service function.

(2) Stub creation procedure

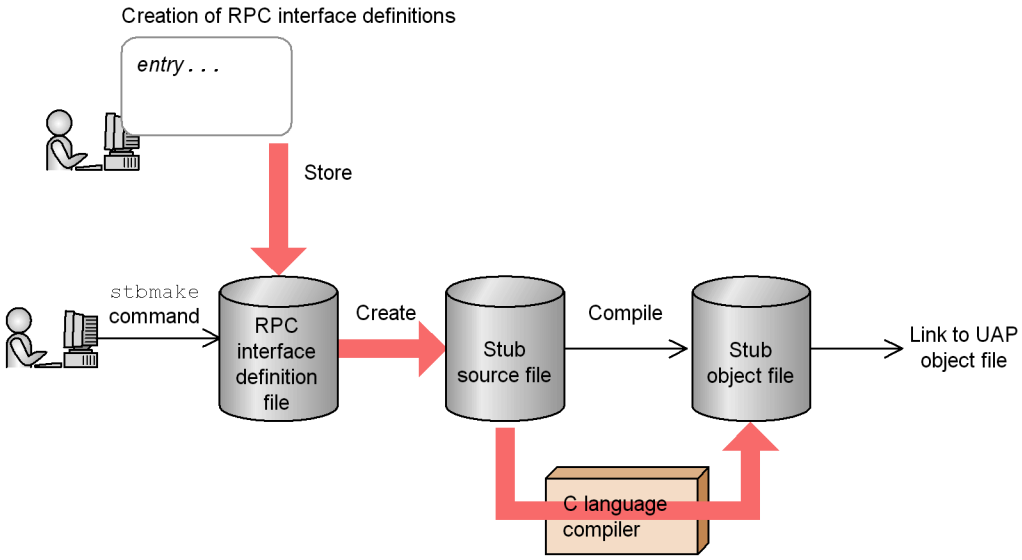
Before creating a stub, create a file (RPC interface definition file) in which UAP service functions are defined. Execute the `stbmake` command with this file as the argument.

When the `stbmake` command is executed, a source file (C-language source file) for the stub is created. Compile this file with the C-language compiler and link it to the object file of the UAP.

When modifying the stub, create the UAP from scratch. Modify the RPC interface definition file, recreate the stub, and link it to the object file of the recompiled UAP.

The figure below shows the stub creation procedure.

Figure 1-7: Stub creation procedure



(3) Creation of RPC interface definition file

When creating a stub, create a file which defines entry points to the SPP and MHP services. This is called the *RPC interface definition*. The file containing this definition is called the *RPC interface definition file*.

Create an RPC interface definition file for each executable file of the SPP or MHP.

(a) Format of RPC interface definition

Write the RPC interface definition in the following format:

Format

```
entry "entry-point-name"["entry-point-name"...];
```

Description

This statement specifies the names of the entry points to the SPP and MHP service functions. Each entry point name must be a C-language function name.

Use 20 characters or fewer to specify each entry point.

The entry point names must correspond to the service names as specified in the user service definition.

Comments can be added to the RPC interface definition. Begin each comment with `/*` and terminate it with `*/`. Comments cannot be nested. Comments cannot be written within a keyword, identifier, or other character string.

More than one `entry` statement can be written in one file. An example of RPC interface definition is given below.

Example

Specification of RPC interface definition for a UAP which has service functions with their entry points identified by `sv01` and `sv02` (use either format below)

Format 1:

```
entry "sv01";
entry "sv02";
```

Format 2:

```
entry "sv01" "sv02";
```

(4) RPC interface definition file name

The file name must end with the suffix `.def` indicating an RPC interface definition file. The directory to contain the file must be in a path that the `stbmake` command can search. No other restrictions are placed on it.

The name of an RPC interface definition file can have up to 255 characters. However, the name that can be specified may be shorter than 255 characters due to OS restrictions.

After the `stbmake` command is executed, a stub source file is created under a name different from that of the RPC interface definition file. Therefore, the RPC interface definition file is not used during the OpenTP1 operation.

1.2.3 Creating stub source file

To create the source file of the stub, execute the `stbmake` command with the RPC interface definition file name as the argument.

(1) File created by stbmake command

When the `stbmake` command is executed, the following file is created (*xxxxx* is the RPC interface definition file name minus the suffix `.def`).

- Stub source file (file name: *xxxxx*`_sstb.c`)

  The name of the source file can be changed using an option to the command.

  The source file name can have up to 255 characters. However, the name that can be specified may be shorter than 255 characters due to OS restrictions. Compile the stub source file with the C-language compiler and link it with the UAP object file.

1.2.4 stbmake - Stub source file creation

(1) Format

```
stbmake [-s [stub-source-file-name]] definition-file-name
```

(2) Description

Creates a stub source file from the RPC interface definition file.

When creating a UAP that uses OpenTP1 remote procedure calls and XATMI interface functions in combination, see the descriptions of the `stbmake` command in *A. Using OpenTP1 Remote Procedure Calls and XATMI-interfaced Functions in Combination*.

(3) Options

- `-s` *stub-source-file-name* ~ <pathname>

  Specify the pathname of the stub source file to be created. If no pathname is specified here, the source file name is the same as the RPC interface definition file name except that the suffix `.def` is replaced with `_sstb.c` and the source file is created in the current directory.

  If a source file with the specified file name is already present, it is replaced with the created source file and is lost.

(4) Command argument

- *definition-file-name*~ <pathname>

  Specify the pathname of the RPC interface definition file.

(5) Notes

The name in the `stbmake` command of a file that can be input and output can be up to 255 characters in length. However, the name that can be specified may be shorter than 255 characters due to OS restrictions.

(6) Example

An example of using the `stbmake` command is given below.

Creating a stub source file from an RPC interface definition file `test.def` in the current directory.

Format 1:

```
stbmake test.def
```

A stub source file `test_sstb.c` is created from an RPC interface definition file `test.def` in the current directory.

Format 2:

```
stbmake -s stub/test.c test.def
```

A directory `stub` is created under the current directory and a stub source file `test.c` is created in the created directory.

1.2.5 Compiling and linking application program

For details on how to compile and link UAPs, see the reference documentation for the OS being used.

- Note on UAP creation

  Be careful of the OpenTP1 version in creating a UAP. Some system services do not accept functions called from UAPs in old versions. To use a UAP created in an old version, the UAP should be recompiled in the OpenTP1 version.

(1) Compilation

To create the object file of a UAP written in C language, compile the source program with the C compiler. Also, use the C compiler to compile the stub source program.

(2) Linkage

The following notes (#1 to #3) apply to files treated in (a) to (d) below.

#1:

The object file for transaction control is required to execute transactions that access the resource manager via the XA interface. Note that any resource manager provided by OpenTP1 is accessed by the XA interface. An object file for transaction control is created by using an OpenTP1 command (`trnmkobj` command). For details on the `trnmkobj` command, see the manual *OpenTP1 Operation*.

#2:

The object file provided by resource manager is required to access the resource manager. The following arguments can be specified in the linkage command to link object files provided by OpenTP1:

Arguments for using the message exchange facility: `-lmcf` and `-lmnet`

Argument for using the DAM access facility: `-ldam`

Argument for using the TAM access facility: `-ltam`

Arguments for using the ISAM facility: `-lismb`, `-lisam`, and `-lrsort`

Argument for using the message queuing facility: `-lmqa`

For details on how to link object files for a non-Hitachi resource manager, see the documentation for the resource manager.

#3:

The object file provided by the online tester is required to use the `dc_uto_test_status` function, which reports the user server test status. The following argument is specified to link the object file for the online tester:

Argument for reporting the user server test status: `-luto`

(a) Files to be linked to SPP and MHP

The executable file of an SPP or MHP is linked to the following files when it is created:

- UAP object file (main and service functions)
- Stub object file
- Object file for transaction control[#1]
- Object file provided by resource manager[#2]
- Object file provided by online tester[#3]
- OpenTP1 library

(b) Files to be linked to SUP

The executable file of an SUP is linked to the following files when it is created:

- UAP object file (main function)
- Object file for transaction control[#1]
- Object file provided by resource manager[#2]
- Object file provided by online tester[#3]
- OpenTP1 library

(c) Files to be linked to UAP that handles offline work

The executable file of UAP that handles offline work is linked to the following files when it is created:

- UAP object file (main function)
- OpenTP1 library

(d) Files to be linked to an SPP or MHP that dynamically loads service functions

When the executable file of an SPP or MHP that dynamically loads service functions is created, it is linked to the following files:

- UAP object file (main function)
- OpenTP1 library
- Object file for transaction control[#1]
- Object file provided by resource manager[#2]
- Object file provided by online tester[#3]

In addition to the above files, the following files are required when the SPP also uses a service search that employs a stub:

- UAP object file (service function)
- Stub object file

(3) Notes

If the OS is HP-UX, always specify `immediate` as the bind mode at linkage. If an executable file created as a bind mode other than `immediate` is used as an OpenTP1 UAP, the system operation is undefined. Use the OS chatr command to check whether the bind mode for the created UAP is `immediate`.

1.3 Creating XATMI interface application programs (TCP/IP, OSI TP)

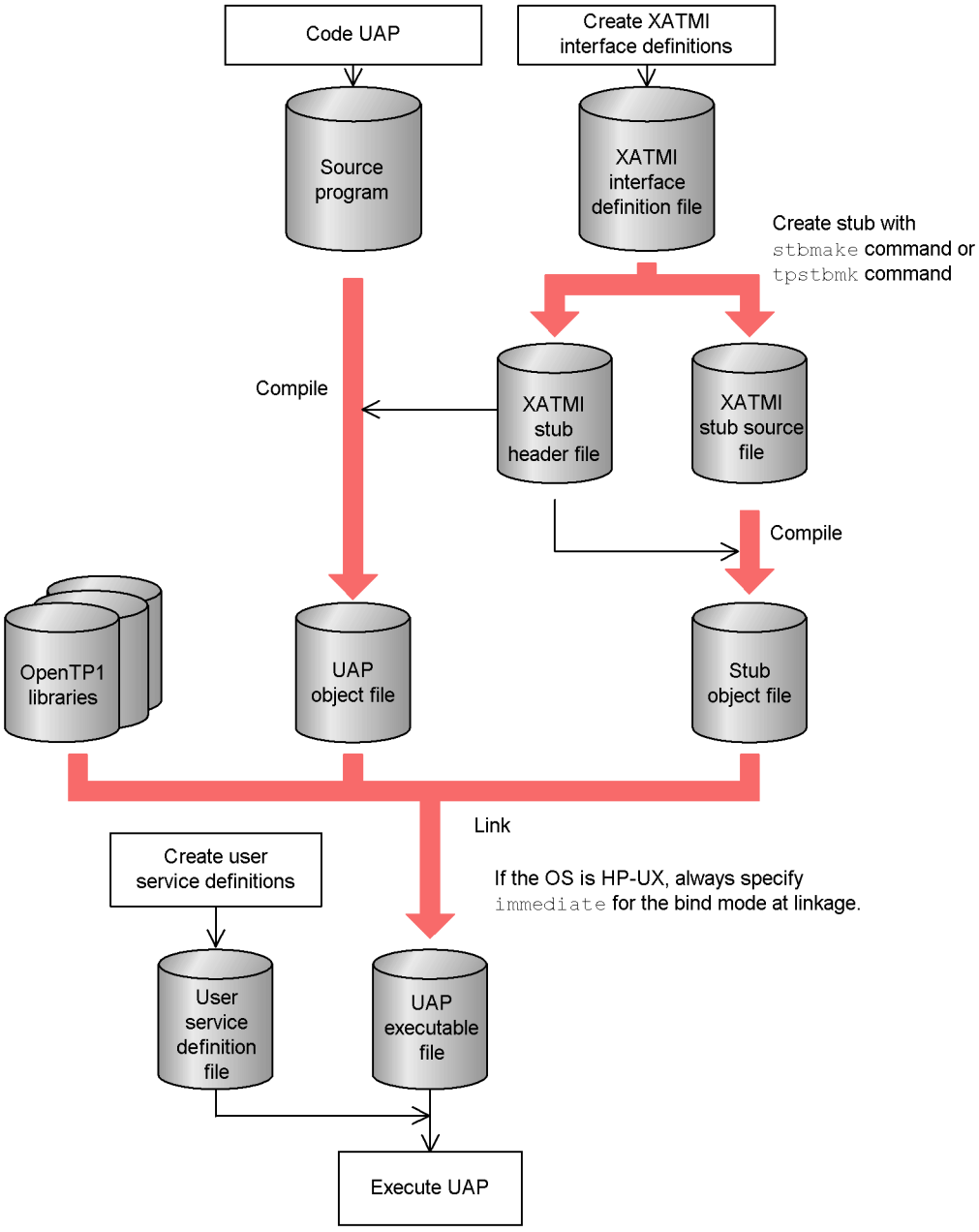
This section explains how to create a UAP that uses an XATMI interface if TCP/IP or OSI TP is used as the communication protocol.

This method differs from how to create a UAP that uses OpenTP1 RPC in terms of the procedure of creating a stub (execution formats for the `stbmake` and `tpstbmk` commands) and in the file to be linked with the UAP. The other procedures are the same as for an OpenTP1 UAP. For details on how to create UAPs, see *1.1 Coding application program* and *1.4 Executing application programs*.

1.3.1 Procedure for creating XATMI-interfaced application programs

The figure below shows the procedure for creating UAP.

Figure 1-8: Procedure for creating UAP (XATMI Interface TCP/IP, OSI TP)



1.3.2 Creating stubs for XATMI interface

This subsection explains how to create the stub for the XATMI interface. For UAP communication through the XATMI interface, stubs are necessary on both the client and server UAPs.

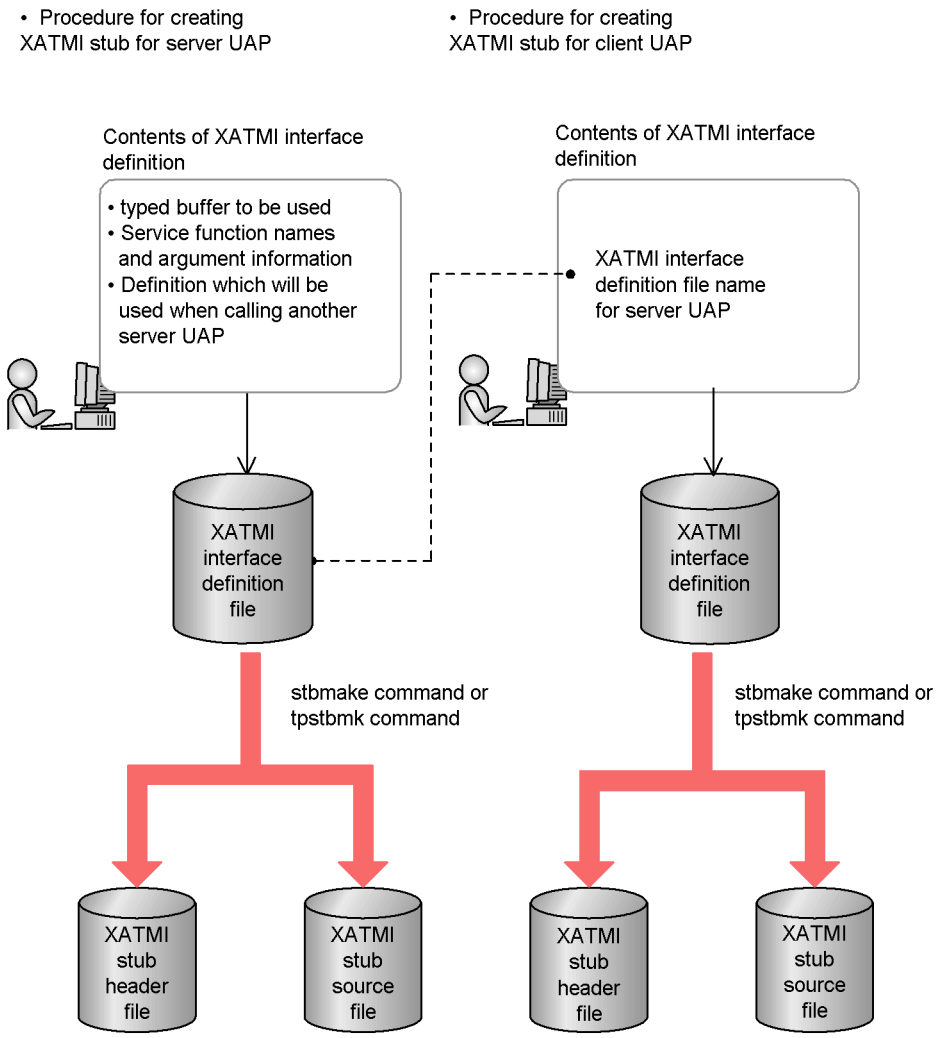
To create a stub, create a file (XATMI interface definition file) that defines an XATMI interface, then execute a stub creating command. The following commands create a stub:

- For a UAP that supports TCP/IP communication: `stbmake` command
- For a UAP that supports OSI TP communication: `tpstbmk` command

Compile the created stub source file with the C-language compiler and link it to the UAP object file.

The figure below provides an overview of the procedure for creating a stub for the XATMI interface.

Figure 1-9: Procedure for creating stub for XATMI interface



(1) XATMI interface definition (for client UAP)

The XATMI interface definition for the client UAP (SUP or SPP) is in the format explained below.

Format

```
called_servers={"server-definition-file-name"
                [,"server-definition-file-name"]...};
```

Description

Specify all XATMI interface definition file names defined in the server UAP. When a server UAP definition file is specified, the typed buffer defined in the server definition file can be used by the client UAP process.

Parameters

- *server-definition-file-name*

  Specify the file name of the XATMI interface definition file of the server UAP. The definition file name must have a suffix `.def`.

  Multiple definition files names can be specified in braces {} in one `called_servers` statement. It is also possible to write multiple `called_servers` statements in one XATMI interface definition file.

Example

Defining a client UAP which communicates with server UAP1 and server UAP2 through the XATMI interface (assuming that the server UAP1 definition file name is `serv1.def` and the server UAP2 definition file name is `serv2.def`).

Format 1:

```
called_servers = { "serv1.def","serv2.def" };
```

Format 2:

```
called_servers = { "serv1.def" };
called_servers = { "serv2.def" };
```

(2) XATMI interface definition (for server UAP)

For the XATMI interface definition of a server UAP, the following items must be specified in any order:

- Definition of the typed buffer to be used
- Definition of service function name and argument information
- `called_servers` statement (if the server UAP is to call another server UAP)

(a) Definition of the typed buffer to be used

Format

```
type-name subtype-name{
        data-type data-name;
      [data-type data-name;]
                  :
                  :
    };
```

Description

Define the type, subtype, and structure of the typed buffer to be used with the server UAP. If the server UAP is to call service from another server UAP process, the typed buffer which can be used by the calling process can also be used by any local process. Therefore, define here only the typed buffer to be used for I/O by the service function within the local process. However, `X_OCTET` will always be recognized. If `X_OCTET` is defined, the execution of a stub creation command (`stbmake` or `tpstbmk` command) will encounter an error.

Parameters

- *type-name*

  Specify the type name of the typed buffer to be used with the server UAP.

- *subtype-name*

  Specify the subtype name of the typed buffer to be used with the server UAP.

- *data-type*

  Specify the data type of the data contained in the structure of the typed buffer to be used with the server UAP.

- *data-name*

  Specify the data name of the data contained in the structure of the typed buffer to be used with the server UAP.

List of the data types that can be used as types

Table 1-6 lists the data types that can be used as *types*. *Identifier* means a data type to be written in the XATMI interface definition. *Data type in C* means data type of a typed buffer actually defined in a stub. To convert a data type in order to communicate with a system other than OpenTP1, specify the identifier to be converted in the XATMI interface definition.

For OpenTP1, a value of type int has four bytes. Therefore, `int4` is written in the definition file so that the fact is explicitly indicated.

Table 1-6: Data types that can be used as types

Type	Identifier	Data type in C	Communication protocol		Remarks
			TCP/IP	OSI TP	
`X_OCTET`	O[#1]	O[#1]	Y	Y	None
`X_COMMON`	short a	short a	Y	Y	None
	short a[n]	short a[n]	Y	Y	None

Type	Identifier	Data type in C	Communication protocol		Remarks
			TCP/IP	OSI TP	
	long a	long a	Y	Y	None
	long a[n]	long a[n]	Y	Y	None
	char a#2	char a	Y	Y	Unconverted array
	octet a	char a	Y	Y	Unconverted array
	tchar a	char a	O	Y	Converted array
	char a[n]#2	char a[n]	Y	Y	Unconverted array
	octet a[n]	char a[n]	Y	Y	Unconverted array
	tchar a[n]	char a[n]	O	Y	Converted array
X_C_TYPE	short a	short a	Y	N	None
	short a[n]	short a[n]	Y	N	None
	long a	DCLONG a	Y	N	None
	long a[n]	DCLONG a[n]	Y	N	None
	int4 a	DCLONG a	Y	N	None
	int4 a[n]	DCLONG a[n]	Y	N	None
	char a#2	char a	Y	N	None
	octet a	char a	Y	N	None
	tchar a	char a	Y	N	None
	char a[n]#2	char a[n]	Y	N	None
	octet a[n]	char a[n]	Y	N	None
	tchar a[n]	char a[n]	Y	N	None
	float a	float a	Y	N	None
	float a[n]	float a[n]	Y	N	None

Type	Identifier	Data type in C	Communication protocol		Remarks
			TCP/IP	OSI TP	
	double a	double a	Y	N	None
	double a[n]	double a[n]	Y	N	None
	octet a[n][n]	char a[n][n]	Y	N	None
	tchar a[n][n]	char a[n][n]	Y	N	None
	str a[n]	char a[n]	Y	N	None
	str a[n][n]	char a[n][n]	Y	N	None
	tstr a[n]	char a[n]	Y	N	None
	tstr a[n][n]	char a[n][n]	Y	N	None

Legend:

Y: Can be used for the applicable communication protocol.

N: Cannot be used for the applicable communication protocol.

O: Even an identifier to be converted is treated as it is without conversion.

#1

`X_OCTET` is automatically recognized if it is not defined. If `X_OCTET` is specified in the `XATMI` interface definition, an error occurs when a command that creates a stub is executed.

#2

This identifier can also be used. However, to create a new program, use one of the following identifiers:

For `X_COMMON`: `octet` or `tchar`

For `X_C_TYPE`: `str` or `tstr`

Example

```
X_C_TYPE subtype1 {
                  char  name[8];
                  int4  data[10];
                  int4  flags;
                  };
```

(b) Definition of service function name and argument information

Format

```
service service-function-name (type-name [subtype-name])|(ALL)|( [void] )};
```

Description

Specify the function name of the service function in the server UAP and the type name and subtype name of the typed buffer to be passed as the arguments. The argument is the data member of the `svc_info` structure which is the actual argument to the service function.

For the `X_OCTET` type, specify only the type name because there is no subtype. If intended processing does not involve reference to the data member of the `svc_info` structure in the service function, assign nothing or void to the argument.

The `tpcall()`, `tpacall()`, and `tpconnect()` functions can call a service function without sending the typed buffer. If data indicated by a member of the svcinfo structure with a service function is not to be referenced explicitly, assign nothing or void to the argument.

To call a specified function, set NULL for the pointer to the typed buffer sent with the `tpcall()`, `tpacall()`, or `tpconnect()` functions at the client side. For the `X_OCTET` type, a specified function can be called even if NULL is not set for the pointer or the length of the sent data is zero.

If specification is not to limit the typed buffer to be received as an argument, assign `ALL` to the argument. The service function defined with argument `ALL` can receive any type of typed buffers as long as they are recognizable in the local process.

Parameters

- *service-function-name*

  Specify the function name in the server UAP.

- *type-name*

  Specify the type name given to the argument to the function.

- *subtype-name*

  Specify the subtype name given to the argument to the function.

Examples

Example 1:

```
service svc_func1(X_C_TYPE subtype1);
```

Example 2 (argument type is X_OCTET):

```
service svc_func2(X_OCTET);
```

Example 3 (service function without argument reception):

```
service svc_func3(void); or service svc_func3();
```

Example 4 (service function without argument limitation):

```
service svc_func4(ALL);
```

(c) If the server UAP is to call another server UAP:

Specify the XATMI interface definition (called_servers statement) of the client UAP.

(3) Name of an XATMI interface definition file

The file name must end with the suffix .def indicating an XATMI interface definition file. The directory to contain the file must be in a path that a stub creation command (stbmake or tpstbmk command) can search. No other restrictions are placed on it.

The name of an XATMI interface definition file can have up to 255 characters. However, the name that can be specified may be shorter than 255 characters due to OS restrictions.

After a command that creates a stub (stbmake or tpstbmk command) is executed, a stub source file is created under a name different from that of the XATMI interface definition file. Therefore, the XATMI interface definition file is not used while OpenTP1 running.

(4) Including the definition file

If the same typed buffer is to be used by different processes, the user can create a definition file for the shared typed buffer and include it in the definition file for each process.

The statement for including the definition file is in the same format as in the C language as follows:

```
#include <file-name> or #include "file-name"
```

The include file will be read through the search path specified by the -i option to a stub creation command (stbmake or tpstbmk command). If the appropriate file is not found in the search path, the current directory will finally be searched.

The file to be included may be given any name (the suffix need not be .h). However, if the file is directly specified in a stub creation command (stbmake or tpstbmk command) as the XATMI interface definition file, observe the definition naming

convention.

The contents of the file to be included are the same as those of the XATMI interface definition file. However, the file should not contain the definition of a service function within the local process in order to avoid name duplication

(5) Naming conventions

1. Service functions and subtypes must be named according to the OpenTP1 rules as follows:
   - Any name cannot begin with `dc`, `DC`, `CBLDC`, `tx`, `TX`, `tp`, or `TP`.
   - Service function names must be 20 characters or less long.
   - The maximum subtype name length is 32 characters. Of these characters, the first 16 characters are valid. These 16 characters are checked for duplication.
   - Up to 32 characters can be used for the data names of data used in the structures of typed buffers.
2. Service function names must be unique within the same process.
3. Subtype names may be duplicate in the same process only if the types and structures are identical. Otherwise, a stub creation command (`stbmake` or `tpstbmk` command) returns with an error.
4. Identical service function names or subtype names may be used in different processes. However, processes treated as different servers will be regarded as the same process by the client if they are called from one client.

1.3.3 Creating stub source files for XATMI interface

Create a stub for the XATMI from the created XATMI interface definition file.

To create a stub, create a file (XATMI interface definition file) that defines an XATMI interface, then execute a stub creation command. The following commands create a stub:

- For a UAP that holds TCP/IP communication: `stbmake` command
- For a UAP that holds OSI TP communication: `tpstbmk` command

Create stubs for the client and server UAPs in the following way:

(1) Files created by the stbmake command or tpstbmk command

The following three files are created by executing the command (*xxxxx* is the `XATMI` interface definition file name minus the suffix `.def`):

- `XATMI` stub source file (default file name: *xxxxx*`_stbx.c`)
- `XATMI` stub header file (default file name: *xxxxx*`_stbx.h`)

- XATMI stub copy file (subtype name followed by .cbl)

The file name can have up to 255 characters. However, the name that can be specified may be shorter than 255 characters due to OS restrictions.

The directory in which a file is created, and the file name can be changed by a command option.

(a) XATMI stub source file

The XATMI stub source file will be compiled with the C-language compiler and linked to the UAP object file.

(b) XATMI stub header file

The XATMI stub header file will be included in the UAP source file and XATMI stub source file.

(c) XATMI stub copy file

The file is used not in a UAP written in C, but rather in a UAP written in COBOL.

1.3.4 stbmake - Stub source file creation for XATMI interface

(1) Format

```
stbmake [-x] [-b] [-S stub-source-file-name]
        [-H stub-header-file-name]
        [-i include-file-pathname]
        [-m server-definition-file-pathname]
        [-p] definition-file-name
```

(2) Description

When you intend to hold TCP/IP communication via an XATMI interface, create the source file for the required XATMI stub. The stbmake command outputs the following files based on the XATMI interface definition file:

- XATMI stub source file
- XATMI stub header file (used in a UAP written in C)
- XATMI stub copy file (used in a UAP written in COBOL)

When creating a UAP that uses OpenTP1 remote procedure calls and XATMI interface functions in combination, see the descriptions of the stbmake command in *A. Using OpenTP1 Remote Procedure Calls and XATMI-interfaced Functions in Combination*.

(3) Options

- -x

  Indicates that the stub created will serve the UAP which uses the XATMI interface. The -x option can be omitted.

- `-b`

  To create an XATMI stub to be used in a UAP in C, omit the `-b` option. To create an XATMI stub copy file to be used in a UAP written in COBOL, specify the `-b` option.

- `-S` *stub-source-file-name* ~ <pathname>

  Specify this option if the XATMI stub source file created is to be renamed. The relative or absolute pathname may be used for this file name.

  If this option is omitted, the file will be created with name *xxxxx*`_stbx.c` in the current directory.

- `-H` *stub-header-file-name* ~ <pathname>

  Specify this option if the XATMI stub header file created is to be renamed. The relative or absolute pathname may be used for this file name.

  If this option is omitted, the file will be created with name *xxxxx*`_stbx.h` in the current directory.

- `-i` *include-file-pathname* ~ <pathname>

  Specify the search path containing the include file specified by the `#include` statement to be used. The `stbmake` command searches the directory identified by the -i option for the include file.

  If the `-i` option is omitted, the current directory is searched for the include file.

  The `-i` option can be specified only once. If more than one search path is needed, the pathnames must be followed by the desired paths separated by colons (:). The search order is the order in which the paths are written as the argument to the `-i` option. Use alphanumeric characters, underscore (_), slash (/), and period (.) when specifying a search pathname.

- `-m` *server-definition-file-pathname* ~ <pathname>

  Specify the search path containing the server definition file to be used. The `stbmake` command searches the directory identified by the `-m` option for the server definition file specified by the `called_servers` statement.

  If the `-m` option is omitted, the current directory is searched for the definition file.

  The `-m` option can be specified only once. If more than one search path is needed, the pathnames must be followed by the desired paths separated by colons (:). The search order is the order in which the paths are written as the argument to the `-m` option.

  Use alphanumeric characters, underscore (_), slash (/), and period (.) when specifying a search pathname.

- `-p`

Specify this option to output the allocation status of the typed buffer in memory to the standard output. Use the `-p` option to learn about how XATMI structure members are allocated in memory.

When the `-p` option is specified, the `stbmake` command creates no files. Thus, output file names specified in the `-S` and `-H` option are ignored. Specify the `-m` and `-i` options to search for files as needed.

(4) Command argument

- *definition-file-name*

  Specify the XATMI interface definition file name. Its suffix must be `.def`.

(5) Notes

- Each option to the `stbmake` command for XATMI stub creation can be specified only once. If an option is specified more than once, the last specified value will be valid
- The name in the `stbmake` command of a file that can be input and output can be up to 255 characters in length. However, the name that can be specified may be shorter than 255 characters due to OS restrictions.

1.3.5 tpstbmk - Creation of an XATMI interface stub OSI TP communication

(1) Format

```
tpstbmk [-b] [-S stub-source-file-name]
        [-H stub-header-file-name]
        [-i include-file-search-pathname]
        [-m server-definition-file-search-pathname]
        definition-file-name
```

(2) Description

When you intend to hold OSI TP communication via an XATMI interface, create the source file for the required XATMI stub. The `tpstbmk` command outputs the following files based on the XATMI interface definition file:

- XATMI stub source file
- XATMI stub header file (used in a UAP written in C)
- XATMI stub copy file (used in a UAP written in COBOL)

When you intend to create a UAP that uses an XATMI interface and OpenTP1 remote procedure calls, seethe explanation about the `tpstbmk` command in *A. Using OpenTP1 Remote Procedure Calls and XATMI-interfaced Functions in Combination*.

(3) Options

- `-b`

  To create an XATMI stub to be used in a UAP in C, omit the -b option. To create an XATMI stub copy file to be used in a UAP written in COBOL, specify the -b option.

- `-S` *stub-source-file-name* ~ <pathname>

  Specify the name of the XATMI stub source file to be created. Relative and absolute pathnames can be used.

  If the `-S` option is omitted, the XATMI stub source file is created in the current directory under the name *XXXXX*`_stbx.c`.

- `-H` *stub-header-file-name* ~ <pathname>

  Specify the name of the XATMI stub header file to be created. Relative and absolute pathnames can be used.

  If the `-H` option is omitted, the XATMI stub header file is created in the current directory under the name *XXXXX*`_stbx.h`.

- `-i` *include-file-search-pathname* ~ <pathname>

  Specify the include file name specified in the `#include` statement of the XATMI interface definition file using a search path. The include file is searched for starting at the directory specified in the `-i` option.

  If the `-i` option is omitted, the search starts at the current directory in which the command was executed.

  The `-i` option can be specified only once. Separate search paths with a colon. The search paths are searched in the order in which they are described in the arguments for the `-i` option.

  Specify a search path using alphanumeric characters, underscore (_), slash (/), and period (.).

- `-m` *server-definition-file-search-pathname* ~ <pathname>

  Specify the server definition file name specified in the `called_servers` statement of the XATMI interface definition file using a search path. The include file is searched for starting at the directory specified in the `-m` option.

  If the `-m` option is omitted, the search starts at the current directory in which the command was executed.

  Specify a search path using alphanumeric characters, underscore (_), slash (/), and period (.).

  The `-m` option can be specified only once. Separate search paths with a colon (:).

The search paths are searched in the order in which they are described in the arguments for the `-m` option.

(4) Command argument

- *definition-file-name* ~ <pathname>

  Specify the name of an XATMI interface definition file. The name must have the suffix `.def`.

(5) Notes

- In the `tpstbmk` command, each option can be specified only once. If an option is specified more than once, only the last value is valid.
- The name in the `tpstbmk` command of a file that can be input and output can be up to 255 characters in length. However, the name that can be specified may be shorter than 255 characters due to OS restrictions.

1.4 Executing application programs

This section explains how to start and terminate UAPs and what environments are needed for executing UAPs.

1.4.1 Starting and terminating each application program

(1) Starting and terminating SUP

(a) Starting

The SUP is started when:

- The OpenTP1 starts if the server name of the SUP is specified in the user service structure definition, or
- The `dcsvstart` command is executed if the server name of the SUP is not specified in the user service structure definition.

Before the SUP can request an SPP for service, the SPP must begin the service and must have started before the SUP has.

(b) Terminating

Once the SUP has been started, it cannot be terminated normally by the OpenTP1. Even when a command to exit the OpenTP1 normally is executed, the OpenTP1 will not terminate until all the SUPs in the OpenTP1 terminate. When coding the SUP, design it so that it will terminate by itself. To bring an SUP into abnormal termination because of some problem, design the SUP so that it will terminate by itself by `exit()` or `abort()`.

The SUP cannot be terminated normally by the `dcsvstop` command. However, the SUP can be brought into forced termination by the `dcsvstop -f` command.

Do not terminate any SUP process by the `kill` command.

(2) Starting and terminating SPP and MHP

(a) Starting

The SPPs and MHPs belonging to one user server (service group) start at once. They start when:

- The OpenTP1 starts if the server name of the SPPs and MHPs is specified in the user service structure definition, or
- The `dcsvstart` command is executed if the server name of the SPPs and MHPs is not specified in the user service structure definition.

If the multiserver facility is in use, the same number of user server processes as the specified number of resident processes are acquired. If the number of service requests

increases, nonresident processes will start as well.

(b) Terminating

The SPP or MHP terminates when:

- Termination processing begins because one of the following OpenTP1 terminate commands is executed:

  `dcstop` (normal termination)

  `dcstop -n` (forced normal termination)

  `dcstop -a` (planned termination A)

  `dcstop -b` (planned termination B)

  `dcstop -f` (forced termination)

- The active online process enters termination steps because one of the following server terminate commands is executed:

  `dcsvstop` (normal termination)

  `dcsvstop -f` (forced termination)

- The active online process is brought into termination by the OpenTP1 because the maximum number of processes in the user service definition is exceeded;
- The SPP or MHP which is executing as a nonresident process finishes service processing; or
- The number of requests addressed to the service group decreases if loads on SPPs or MHPs are distributed using a multiserver configuration.

Do not terminate any SPP or MHP process by the `kill` command.

(3) Starting and terminating UAPs that handle offline work

Users can start UAPs that handle offline work by any method. The UAPs are terminated by terminating the processes by the shell. Users are responsible for starting and terminating UAPs that handle offline works.

1.4.2 Operating environment of application programs started by OpenTP1

- The standard input (`stdin`), standard output (`stdout`), and standard error output (`stderr`) of SUPs, SPPs, and MHPs are redirected by OpenTP1.
- When a UAP is activated, a directory `$DCDIR/tmp/home/`*user-server-name.xx* (where *xx* is a sequence number) is created. The UAP runs with this directory as the current working directory.

  You can change this directory by setting the `prc_current_work_path` operand

in the system common definition.

- The user ID (UID) and group ID (GID) have the values specified at environment setup for the user server.
- The root directory remains a forward slash (/).
- The following file descriptors are open during UAP execution:

  File descriptor 0: Standard input file descriptor

  File descriptor 1: Standard output file descriptor

  File descriptor 2: Standard error output file descriptor
- `umask` is `000`.
- No control terminal is used.
- OpenTP1 automatically sets a UAP signal when a UAP process is created. The table below lists UAP signals set by OpenTP1.

  Table 1-7: UAP signals set by OpenTP1

Signal name	Setting upon UAP process creation	Operation
SIGHUP	SIG_DFL(default)	exit
SIGINT	SIG_IGN(ignored)	ignore
SIGQUIT	SIG_DFL(default)	core
SIGILL	SIG_DFL(default)	core
SIGTRAP	SIG_IGN(ignored)	ignore
SIGIOT#	SIG_DFL(default)	core
SIGABRT#	SIG_DFL(default)	core
SIGEMT	SIG_DFL(default)	core
SIGFEP	SIG_DFL(default)	core
SIGKILL	-	exit
SIGBUS	SIG_DFL(default)	core
SIGSEGV	SIG_DFL(default)	core
SIGSYS	SIG_DFL(default)	core
SIGPIPE#	SIG_IGN(ignored)	ignore
SIGALRM	SIG_IGN(ignored)	ignore

Signal name	Setting upon UAP process creation	Operation
SIGTERM	SIG_DFL(default)	exit
SIGUSR1	SIG_IGN(ignored)	ignore
SIGUSR2	SIG_IGN(ignored)	ignore
SIGCLD	SIG_DFL(default)	ignore

Legend:

N: Not applicable

Note

When specifying signal operations using UAP, do not stop the process by invoking exit() or abort() within the specified signal handler. When the process is stopped in the signal handler, the OpenTP1 system will shut down even if the signal interruption occurs during critical OpenTP1 processing. Furthermore, do not rewrite the value of the external variable errno in the signal handler.

#

The signals marked with # cannot be respecified. Do not change the settings of these signal operations in the program when creating a UAP.

1.4.3 Application's environment variables

UAP environment variables can be set for each user server at environment setup for the user server. However, the following environment variables are set by OpenTP1.

The OpenTP1 sets the following environment variables:

- DCDIR: OpenTP1 home directory
- DCCONFPATH: Directory containing OpenTP1 system definition files
- DCSVNAME: User server name
- DCSVGNAME: Service group name (can be referenced only with SPPs or MHPs)
- DCUAPCONFPATH: Directory containing OpenTP1 user service definition files (only when the files are to be stored in a different directory from DCCONFPATH)

In addition to the above, environment variables beginning with DC are used by the OpenTP1. Since these environment variables are for reference only, do not change them. If changed, the system operation is undefined.

SUPs, SPPs, and MHPs that run under OpenTP1 do not inherit the environment variables set when the user logs in as an OpenTP1 system administrator using telnet or other means. Set these environment variables again in the user service definition.

Chapter

2. Syntax of OpenTP1 Library Functions

This chapter explains the syntax of OpenTP1 library functions.

This chapter contains the following sections:

Format for explaining functions

This section explains functions provided by OpenTP1 in the following format:

Format

Indicates the formats of OpenTP1 library functions and the data types of arguments.

To code a UAP in *C++ language* or the *ANSI C* format, see the format provided under *ANSI C, C++* in the function's *Format* section. To code a UAP in the pre-ANSI K&R format, see the format provided under *K&R C* in the function's *Format* section.

Use the data types given in this section when allocating values to arguments. A specific name can be arbitrarily assigned to an argument if not specially noted.

Description

Explains the facilities of the corresponding function.

Argument(s) whose value(s) is set in the UAP

Indicates the argument(s) whose value(s) should be specified when the function is executed. Specify a value for each argument according to the explanation. If a value is not always specified for an argument, the explanation of the argument is enclosed in brackets [ ] when the value is specified for the argument.

Argument(s) whose value(s) is returned from OpenTP1

Indicates the argument(s) whose value(s) is returned from OpenTP1 after the function is executed. Reference the contents of the argument after the function is executed. If a value is not always returned to an argument from OpenTP1, the explanation of the argument is enclosed in brackets [ ] when the value is returned.

Argument(s) whose value(s) is passed from a client UAP

Indicates the argument(s) whose value(s) is passed from the client UAP when the service function is used. Execute service function processing referencing the contents of the argument.

Argument(s) whose value(s) is returned from a server UAP

Indicates the argument(s) whose a value(s) is returned from the service function when a synchronous-response-type RPC or asynchronous-response-type RPC is used. The UAP that called the function `dc_rpc_call()` or the function `dc_rpc_poll_any_replies()` can reference the value of the argument shown here.

Return values

Values returned when the function is executed are explained in a table. The return

value indicates whether the function was executed normally. If an error occurs, the return value indicates the error status.

To maintain interchangeability, use the return value with the constant name shown here when creating a UAP. The constant name of the return value is defined in the header file. Reference the header file definition when you need the information of the return value.

Example

Provided only for functions with which examples are necessary

Note(s)

Explains a note(s) on using the function.

Creating main and service functions

This section gives the syntax and other information of the following OpenTP1 UAP main and service functions. The SPP and MHP create main and service functions, whereas the SUP creates only main functions.

- Create a main function (SUP, SPP, MHP)
- Create a service function (SPP)
- Create a service function (MHP)

The method for creating SGW main and service functions must conform to the specification of the open system being used.

TP1/LiNK can use only the SUP, SPP and MHP as the OpenTP1 UAP. However, TP1/Messaging is required when you create MHPs under TP1/LiNK.

Create a main function (SUP, SPP, MHP)

Format

The name of a main function must include main(). For the other rules of creating main functions, comply with the specifications of the C language for coding. OpenTP1 does not limit creation of main functions. Main functions can be created according to the explanation of this section.

Description

After the UAP process starts, the OS first calls the main function.

- SUP main function

  The following OpenTP1 functions are always called in the SUP main function:

  1. `dc_rpc_open()` (Start an application program)
  2. `dc_adm_complete()` (Report the completion of user server start processing)
  3. `dc_rpc_close()` (Terminate an application program after job terminate)

  In addition to the above OpenTP1 functions, the function for initializing UAP processes required for jobs, the termination processing function, and the function `dc_rpc_call()` can also be called.

- SPP main function

  Service functions created as services which are provided by an SPP are grouped into one executable file. An executable file comprising one main function and multiple service functions corresponds to a service group.

  The OpenTP1 functions listed below are always called in the SPP main function. To use an MCF function with an SPP service, call the function `dc_mcf_open()` and the function `dc_mcf_close()`.

  1. `dc_rpc_open()` (Start an application program)
  2. `dc_rpc_mainloop()` (Start an SPP service)
  3. `dc_rpc_close()` (Terminate an application program after job terminate)

  After initialization processing, the main function stops when the function `dc_rpc_mainloop()` is called. Meanwhile, the main function performs processing requested by service functions. In addition to the above OpenTP1 functions, the function for initializing SPP processes required for jobs, the termination processing function, and the function `dc_rpc_call()` can also be used in the main function.

- MHP main function

  Service functions created as applications for message processing are grouped into one executable file. An executable file comprising one main function and multiple service functions corresponds to a service group. The service group name must be unique in the domain (in the entire network).

  The following OpenTP1 functions are always called in the MHP main function:

  1. `dc_rpc_open()` (Start an application program)
  2. `dc_mcf_open()` (Open the MCF environment)
  3. `dc_mcf_mainloop()` (Start an MHP service)
  4. `dc_mcf_close()` (Close the MCF environment)
  5. `dc_rpc_close()` (Terminate an application program after job terminate)

  The MHP having the service function corresponding to the application name is started. After initialization processing, the main function stops when the function `dc_mcf_mainloop()` is called. Meanwhile, the main function performs processing requested by service functions. In addition to the above OpenTP1 functions, the function for initializing MHP processes required for jobs, the termination processing function, and the function `dc_rpc_call()` can also be used in the main function.

Argument

No argument is passed to the main function.

Create a service function (SPP)

Format

■ ANSI C, C++

```
void function-name (char *in, DCULONG *in_len, char *out,
                    DCULONG *out_len)
 {
  Service processing
  }
```

■ K&R C

```
void function-name (in, in_len, out, out_len)
char               *in;
DCULONG            *in_len;
char               *out;
DCULONG            *out_len;
{
Service processing
}
```

Description

The SPP service function executes a service and returns the execution results. The SPP service function is called by the function `dc_rpc_call()` of the client UAP. Create the service function in the above format as required.

The service function name corresponds to the entry point name of the service function. Specify this correspondence at execution environment setup for a UAP. The method of execution environment setup for a UAP is as follows:

- For TP1/Server Base, specify the correspondence in the user service definition.
- For TP1/LiNK, execute a command for setting up an environment for a UAP to specify the correspondence interactively.

Argument specification

The values listed below are passed as arguments to the service function. These values are specified in the function `dc_rpc_call()` of the client UAP.

- Input parameter (`in`)
- Input parameter length (`in_len`)
- Response length (`out_len`)

  The values specified for the input parameter and input parameter length in the client UAP are passed to the service function as they are. (The expression formats

of character codes and numbers are not converted.) The length specified in the client UAP is passed as the response length.

For the service function, set the following values for arguments:

- Service function response (`out`)
- Length of the service function response (`out_len`)

  Set a response for out, set the response length for `out_len`, then return the service function.

A response is sent to the service client UAP regardless of whether the service function was executed as a transaction or whether commitment or rollback processing was executed. Create a response with which the service function informs the client UAP of the occurrence of an error if necessary.

Arguments whose values are passed from the client UAP

- `in`

  The input parameter specified in the client UAP is passed.

- `in_len`

  The input parameter length specified in the client UAP is passed.

- `out_len`

  The response length specified in the client UAP is passed.

Arguments whose values are set in the UAP

- `out`

  Specify the response from the service function. Return the service function after specifying the processing results for out.

- `out_len`

  Specify the length of the actual response from the service function. Set a numeric value which is equal to or smaller than the `out_len` value passed from the client UAP.

Notes on service function processing

1. The service function called by the function `dc_rpc_call()` of an nonresponse-type RPC (`DCRPC_NOREPLY` specified for flags) cannot reference out and `out_len`.
2. If the service function is written in C language, the value upon the previous service request remains in the static variable. Thus, initialize the value before using it if necessary.
3. The following functions cannot be used from the service function:

- The function dc_rpc_open(), the function dc_rpc_close(), and the function dc_rpc_mainloop() cannot be called. Also, do not use exit() in the service function. The UAP operation is not ensured if any of the functions or exit() is used.
- After system calls such as fork(), exec(), and system() are called to create a child process, all the OpenTP1 functions cannot be called from the child process.

4. Before an SPP service function can call a message exchange function (dc_mcf_~), the main function must call the functions dc_mcf_open() and dc_mcf_close().
5. The function dc_mcf_receive() cannot be called from SPP service functions.
6. Do not execute an operation or reference that extends beyond the area of the input parameter length passed to in_len, for the input parameter passed to in. If you execute such an operation or reference, operation cannot be guaranteed. The process may terminate abnormally.

Relationship between transactions and the service function

The service function is executed as a transaction branch upon the request of a service in the following case:

- The transaction attribute has been specified in the user service definition of the process that executes the service function, and the client UAP has been executed as a transaction.

In the above case, do not use the function dc_trn_begin() in the service function.

Commitment or rollback processing is ensured for all global transaction services. When the service function operating as a transaction branch issues return, the service function is assumed to request normal termination of the transaction branch.

The service function is not executed as a transaction in the following case:

- The transaction attribute has been specified in the user service definition, but the client UAP has not been executed as a transaction.

To execute the service function as a transaction, use the function dc_trn_begin() and the function dc_trn_unchained_commit() from the service function at any time in order to start the transaction and acquire a synchronization point.

When no transaction attribute is specified in the user service definition, the service function cannot be executed as a transaction by using the function dc_trn_begin() from the service function.

Return value

No return value. The value specified with return() is not returned to the client UAP.

OpenTP1 does not also reference any return value. Specifying `-1` as a return value does not request rollback processing.

Create a service function (MHP)

Format

■ ANSI C, C++

```
void function-name (void)
{
Service processing
}
```

■ K&R C

```
void function-name ()
{
Service processing
}
```

Description

The MHP service function executes a service and returns the execution results. When the MCF receives a message, the MHP having the service function that corresponds to the application name is started.

Create the MHP service function in the above format as required. The service function name corresponds to the entry point name of the service function. Specify this correspondence in the user service definition of the process that executes the service function.

The correspondence between the service name and the application name is specified in the MCF application definition.

Argument

None

Notes on service function processing

1. The following functions cannot be called from the service function:

`dc_rpc_open()`

`dc_rpc_close()`

`dc_mcf_open()`

`dc_mcf_close()`

`dc_rpc_mainloop()`

`dc_mcf_mainloop()`

Also, do not use `exit()` in the service function. The UAP operation is not ensured if any of the functions or `exit()` is used

1. After system calls such as `fork()`, `exec()`, and `system()` are called to create a child process, all the OpenTP1 functions cannot be called from the child process.
2. Another UAP cannot use a service request to the MHP service function by using the function `dc_rpc_call()`.

Return value

No return value. Specifying `-1` as a return value does not request rollback processing.

System operation management (dc_adm_~)

This section gives the syntax and other information of the following functions which are called by UAPs and use various OpenTP1 system facilities:

- `dc_adm_call_command` - Execute an operation command
- `dc_adm_complete` - Report the completion of user server start processing
- `dc_adm_status` - Report the status of a user server

The functions for system operation management (`dc_adm_~`) can be used in UAPs of both TP1/Server Base and TP1/LiNK.

dc_adm_call_command - Execute an operation command

Format

- ANSI C, C++

```
#include <dcadm.h>
int dc_adm_call_command (char *com, int *stat,
                         char *outmsg, DCULONG *outsiz,
                         char *errmsg, DCULONG *errsiz,
                         DCLONG flags)
```

- K&R C

```
#include <dcadm.h>
int dc_adm_call_command (com, stat, outmsg, outsiz,
                         errmsg,errsiz, flags)
char      *com;
int       *stat;
char      *outmsg;
DCULONG   *outsiz;
char      *errmsg;
DCULONG   *errsiz;
DCLONG     flags;
```

Description

The function `dc_adm_call_command()` passes com from the UAP to sh(1) as in the case of command entry in online mode. The process waits until the shell completes its processing, and returns the exit status of the shell. After command processing is completed, the standard output information and the standard error output information are returned.

If the OpenTP1 uses UAPs which execute operation commands, add the directory containing the commands to the search path. Use any of the following methods for addition to the search path.

- Specify the path name of the command in the `prcsvpath` operand of the process service definition.
- Add the search path with the `prcpath` command.
- Assign putenv PATH to environment variable in the user service definition.

Arguments whose values are set in the UAP

- `com`

  Specify the character string of the operation command to be executed.

- outsiz

  The execution results of the operation command are output to the standard output file. Specify the size of the contents (value returned to outmsg) in bytes. Pre-allocate the area in size of the number of bytes that is to be specified for outsiz. The area must begin from the address pointed to by outmsg. The number of bytes to be specified for this argument must be decided according to the command executed by the UAP.

  After processing terminates, the actual length that was output as the execution results of the command to the standard output file is returned.

- errsiz

  The execution results of the command are output to the standard error output file. Specify the size of the contents (value returned to errmsg) in bytes. Pre-allocate the area in size of the number of bytes that is to be specified for errsiz. The area must begin from the address pointed to by errmsg. The number of bytes to be specified for this argument must be decided according to the command executed by the UAP.

  After processing terminates, the actual length that was output as the execution results of the command to the standard error output file is returned.

- flags

  Specify the operation of the function dc_adm_call_command() if the complete data of a standard output message or standard error output message cannot be acquired.

  DCADM_DELAY

  Processing is stopped by canceling the processing for the executed command.

  DCNOFLAGS

  Only acquired data is returned to the argument, and the function returns with an error.

Arguments whose values are returned from OpenTP1

- stat

  A shell termination code# is returned indicating whether the specified command terminated normally or abnormally.

  #: Denotes an sh(1) termination status in the format specified by waitpid(2).

- outmsg

  The character string that was output as the execution results of the command to the standard output file is returned. The maximum number of bytes for the character string is (outsiz-1). If the character string exceeds the maximum number of bytes (outsiz-1), the excess characters are truncated. If the character string exceeds the capacity of the pipe, the excess characters are also truncated. If the character string

does not reach the maximum number of bytes (outsiz-1), the entire character string is returned. A null character is suffixed to the character string to be stored.

- outsiz

  The length of the character string that was output as the execution results of the command to the standard output file is returned.

- errmsg

  The character string that was output as the execution results of the command to the standard error output file is returned. The maximum number of bytes for the character string is (errsiz-1). If the character string exceeds the maximum number of bytes (errsiz-1), the excess characters are truncated. If the character string exceeds the capacity of the pipe, the excess characters are also truncated. If the character string does not reach the maximum number of bytes (errsiz-1), the entire character string is returned. A null character is suffixed to the character string to be stored.

- errsiz

  The length of the character string that was output as the execution results of the command to the standard error output file is returned.

Return values

Return value	Return value (numeric)	Explanation
DC_OK	0	The shell termination code is 0 (normal termination of the command execution). The character string was stored in the standard output area and the standard error output area.
DCADMER_STATNOTZERO	-1855	The shell termination code is not 0 (abnormal termination of the command execution). Standard output data and standard error output data were stored in the areas.
DCADMER_PARAM	-1852	The argument value is invalid.
DCADMER_MEMORY_OUT	-1856	All the standard output data could not be stored in the area.
DCADMER_MEMORY_ERR	-1857	All the standard error output data could not be stored in the area.
DCADMER_MEMORY_OUTERR	-1858	Both the standard output data and the standard error output data could not be stored in the areas.
DCADMER_SYSTEMCALL	-1859	A system call (close, pipe, dup, or read) could not be executed.

Note

Be careful not to duplicate the command name between directories that are specified as search paths. The correct command will not execute if the command name is duplicated. In addition, be careful not to duplicate the command name with that of the command group provided by OpenTP1 (under `$DCDIR/bin`).

dc_adm_complete - Report the completion of user server start processing

Format

■ ANSI C, C++

```
#include <dcadm.h>
int dc_adm_complete (DCLONG flags)
```

■ K&R C

```
#include <dcadm.h>
int dc_adm_complete (flags)
DCLONG      flags;
```

Description

This function `dc_adm_complete()` notifies the OpenTP1 that SUP activation has been completed. SUP activation is completed when the function `dc_adm_complete()` normally returns.

SPPs and MHPs assume the completion of start processing when the function `dc_rpc_mainloop()` or the function `dc_mcf_mainloop()` terminates normally. Thus, there is no need to call the function `dc_adm_complete()` for SPPs and MHPs.

The function `dc_adm_complete()` cannot be called from UAP that handles offline work.

Argument whose value is set in the UAP

■ `flags`

Specify `DCNOFLAGS`.

Return values

Return value	Return value (numeric)	Explanation
`DC_OK`	`0`	Normal termination.
`DCADMER_COMM`	`-1851`	An error occurred during communication between processes.
`DCADMER_PARAM`	`-1852`	The argument value is invalid.
`DCADMER_STS_IO`	`-1853`	A status information input/output error occurred.

Return value	Return value (numeric)	Explanation
DCADMER_PROTO	-1854	The user server is not being started/restarted normally, or the function dc_rpc_open() was not called.

dc_adm_status - Report the status of a user server

Format

- ANSI C, C++

```
#include <dcadm.h>
int dc_adm_status (DCLONG flags)
```

- K&R C

```
#include <dcadm.h>
int dc_adm_status (flags)
DCLONG      flags;
```

Description

The function `dc_adm_status()` reports the status of the user server that called the function. The user server status is reported with the return value.

Argument whose value is set in the UAP

- `flags`

  Specify `DCNOFLAGS`.

Return values

When the return value is positive (indicating the user server status):

Return value	Return value (numeric)	Explanation
DCADM_STAT_START_NORMAL	2	The user server is being started normally.
DCADM_STAT_START_RECOVER	3	The user server is being restarted normally.
DCADM_STAT_ONLINE	4	The user server is in online mode.
DCADM_STAT_STOP	5	The user server is being terminated.

When the return value is negative (indicating an error):

Return value	Return value (numeric)	Explanation
DCADMER_COMM	-1851	An error occurred during communication between processes.
DCADMER_PARAM	-1852	The argument value is invalid.

Return value	Return value (numeric)	Explanation
DCADMER_STS_IO	-1853	A status information input/output error occurred.
DCADMER_PROTO	-1854	The function dc_adm_status() was called from a UAP that handles offline work. The function dc_adm_status() cannot be used with UAP that handles offline work.
		The function dc_rpc_open() was not called.

Multinode facility (dc_adm_get_~)

This section gives the syntax and other information of the following functions which are used for multinode facilities:

- `dc_adm_get_nd_status` - Acquire the status of a specified OpenTP1 node
- `dc_adm_get_nd_status_begin` - Start acquiring the status of an OpenTP1 node
- `dc_adm_get_nd_status_done` - Terminate acquiring the status of an OpenTP1 node
- `dc_adm_get_nd_status_next` - Acquire the status of an OpenTP1 node
- `dc_adm_get_nodeconf_begin` - Start acquiring a node identifier
- `dc_adm_get_nodeconf_done` - Terminate acquiring a node identifier
- `dc_adm_get_nodeconf_next` - Acquire a node identifier
- `dc_adm_get_node_id` - Acquire the node identifier of the local node
- `dc_adm_get_sv_status` - Acquire the status of a specified user server
- `dc_adm_get_sv_status_begin` - Start acquiring the status of a user server
- `dc_adm_get_sv_status_done` - Terminate acquiring the status of a user server
- `dc_adm_get_sv_status_next` - Acquire the status of a user server

The functions for multinode facility (`dc_adm_get_~`) can be used only in UAPs of TP1/Server Base. They cannot be used in UAPs of TP1/LiNK.

dc_adm_get_nd_status - Acquire the status of a specified OpenTP1 node

Format

- ANSI C, C++

```
#include <dcadm.h>
int dc_adm_get_nd_status (char *node_id, DCLONG flags)
```

- K&R C

```
#include <dcadm.h>
int dc_adm_get_nd_status (node_id, flags)
char       *node_id;
DCLONG      flags;
```

Description

The function `dc_adm_get_nd_status()` acquires the status of a specified OpenTP1 node.

This function acquires the status of the execution system when the function `dc_adm_get_nd_status()` is called with a specified OpenTP1 node for the system switch configuration.

Arguments whose value is set in the UAP

- `node_id`

  Specify the pointer to the node identifier. Add a null character after the node identifier.

  The length of the node identifier must be equal to the length defined by `DCADM_NODE_ID_LEN`. If a node identifier with a different length is specified, the function returns with an error.

- `flags`

  Specify `DCNOFLAGS`.

Return values

When the return value is positive (indicating the OpenTP1 node status):

Return value	Return value (numeric)	Explanation
DCADM_STAT_NOT_UP	9	Communication with the specified OpenTP1 node is impossible for the following reason: • The OpenTP1 at the OpenTP1 node must be defined or redefined with the dcsetup command • The value specified in the multinode physical definition is incorrect (the OpenTP1 node is not defined or the specified host name or port number is incorrect). • A communication error occurred (power is not supplied to the OpenTP1 node machine or a network error occurred).
DCADM_STAT_TERM	8	The OpenTP1 node is halted or is being terminated abnormally.
DCADM_STAT_START_NORMAL	2	The OpenTP1 node is normally being started.
DCADM_STAT_START_RECOVER	3	The OpenTP1 node is normally being restarted.
DCADM_STAT_ONLINE	4	The OpenTP1 node is online.
DCADM_STAT_STOP	5	The OpenTP1 node is normally being terminated.
DCADM_STAT_STOPA	6	The OpenTP1 node is being terminated according to plan A.
DCADM_STAT_STOPB	7	The OpenTP1 node is being terminated according to plan B.
DCADM_STAT_SWAP	10	The system is being switched.

When the return value is negative (indicating an error):

Return value	Return value (numeric)	Explanation
DCADMER_COMM	-1851	An inter-process communication error occurred.
DCADMER_PARAM	-1852	The value specified for the argument is invalid.
DCADMER_PROTO	-1854	The function dc_rpc_open() was not called.
DCADMER_MEMORY	-1861	The memory became insufficient.
DCADMER_DEF	-1862	An incorrect value is specified in the multinode configuration definition or in the multinode physical definition.

Return value	Return value (numeric)	Explanation
DCADMER_MULTI_DEF	-1864	N is specified for multi_node_option in the system common definition.
		The TP1/Multi is not installed in the system.
		The correct version TP1/Multi is not installed in the system.
DCADMER_REMOTE	-1866	The specified OpenTP1 node cannot use the multinode facility for the following reason: • N is specified for multi_node_option in the system common definition. • The TP1/Multi is not installed in the system. • The correct version TP1/Multi is not installed in the system. • The memory became insufficient.
DCADMER_NODE_NOT_EXIST	-1867	The node identified by node_id is not included in the OpenTP1 nodes.

dc_adm_get_nd_status_begin - Start acquiring the status of an OpenTP1 node

Format

- ANSI C, C++

```
#include <dcadm.h>
int dc_adm_get_nd_status_begin (char *sub_area,
                                DCLONG *entry_count,
                                DCLONG flags)
```

- K&R C

```
#include <dcadm.h>
int dc_adm_get_nd_status_begin (sub_area, entry_count,
                                flags)
char       *sub_area;
DCLONG     *entry_count;
DCLONG      flags;
```

Description

The function dc_adm_get_nd_status_begin() starts acquiring the status of an OpenTP1 node. When this function terminates normally, it returns the number of OpenTP1 nodes whose status will be acquired.

Arguments whose value is set in the UAP

- sub_area

  Specify the pointer to the multinode subarea identifier or character string (*). Add a null character after the multinode subarea identifier. If the pointer to the character string (*) is specified, the function will acquire the statuses of all OpenTP1 nodes making up the multinode area.

  The length of the multinode subarea identifier must be equal to or less than the maximum length defined by DCADM_SUB_AREA_NAME_SIZE. If a longer identifier is specified, the function returns with an error.

- entry_count

  Specify the pointer to the area to which the number of OpenTP1 nodes will be returned. The area set here will contain the number of OpenTP1 nodes in the multinode subarea identified by sub_area. If the pointer to the character string (*) is specified for sub_area, the number of all OpenTP1 nodes in the multinode area will returned to the area.

- flags

  Specify DCNOFLAGS.

Return values

Return value	Return value (numeric)	Explanation
DC_OK	0	Normal termination. This value is returned even if the specified multinode subarea contains an OpenTP1 involving a communication error.
DCADMER_PARAM	-1852	The value specified for the argument is invalid.
DCADMER_SUBAREA_NOT_EXIST	-1860	There is no multinode subarea with the name specified for sub_area.
DCADMER_MEMORY	-1861	The memory became insufficient.
DCADMER_DEF	-1862	An incorrect value is specified in the multinode configuration definition or in the multinode physical definition.
DCADMER_PROTO	-1854	The function dc_adm_get_nd_status_begin() was already called.
		The function dc_rpc_open() was not called.
DCADMER_MULTI_DEF	-1864	N is specified for multi_node_option in the system common definition.
		The TP1/Multi is not installed in the system.
		The correct version TP1/Multi is not installed in the system.

dc_adm_get_nd_status_done - Terminate acquiring the status of an OpenTP1 node

Format

■ ANSI C, C++

```
#include <dcadm.h>
int dc_adm_get_nd_status_done (DCLONG flags)
```

■ K&R C

```
#include <dcadm.h>
int dc_adm_get_nd_status_done (flags)
DCLONG   flags;
```

Description

The function `dc_adm_get_nd_status_done()` terminates acquiring the status of an OpenTP1 node. Call this function when the return value from the function `dc_adm_get_nd_status_begin()` is `DC_OK`.

Arguments whose value is set in the UAP

■ `flags`

Specify `DCNOFLAGS`.

Return values

Return value	Return value (numeric)	Explanation
`DC_OK`	0	Normal termination.
`DCADMER_PARAM`	-1852	The value specified for the argument is invalid.
`DCADMER_PROTO`	-1854	The function `dc_adm_get_nd_status_begin()` was not called.
		The function `dc_rpc_open()` was not called.
`DCADMER_MULTI_DEF`	-1864	`N` is specified for `multi_node_option` in the system common definition.
		The TP1/Multi is not installed in the system.
		The correct version TP1/Multi is not installed in the system.

dc_adm_get_nd_status_next - Acquire the status of an OpenTP1 node

Format

- ANSI C, C++

```
#include <dcadm.h>
int dc_adm_get_nd_status_next (char *node_id,
                               DCLONG flags)
```

- K&R C

```
#include <dcadm.h>
int dc_adm_get_nd_status_next (node_id, flags)
char       *node_id;
DCLONG      flags;
```

Description

The function `dc_adm_get_nd_status_next()` acquires the status of one OpenTP1 node in the multinode area containing the user server which has called this function or of one OpenTP1 node in a specified multinode subarea.

This function acquires the status of the execution system when the function `dc_adm_get_nd_status_next()` is called with a specified OpenTP1 node for the system switch configuration.

The OpenTP1 node status as acquired by this function is the status which stood when the function `dc_adm_get_nd_status_begin()` was called.

Arguments whose value is set in the UAP

- `node_id`

  Specify the pointer to the area which will receive the node identifier of the OpenTP1 node. A null character is added at the end of the node identifier. The length of the area must be equal to the length defined by `DCADM_NODE_ID_SIZE`.

- `flags`

  Specify `DCNOFLAGS`.

Return values

When the return value is positive (indicating the OpenTP1 node status):

Return value	Return value (numeric)	Explanation
DCADMER_STAT_NOT_UP	9	Communication with the specified OpenTP1 node is impossible for the following reason: • The OpenTP1 at the OpenTP1 node must be defined or redefined with the dcsetup command. • The value specified in the multinode physical definition is incorrect (the OpenTP1 node is not defined or the specified host name or port number is incorrect). • A communication error occurred (power is not supplied to the OpenTP1 node machine or a network error occurred).
DCADM_STAT_TERM	8	The OpenTP1 node is halted or is being terminated abnormally.
DCADM_STAT_START_NORMAL	2	The OpenTP1 node is normally being started.
DCADM_STAT_START_RECOVER	3	The OpenTP1 node is normally being restarted.
DCADM_STAT_ONLINE	4	The OpenTP1 node is online.
DCADM_STAT_STOP	5	The OpenTP1 node is normally being terminated.
DCADM_STAT_STOPA	6	The OpenTP1 node is being terminated according to plan A.
DCADM_STAT_STOPB	7	The OpenTP1 node is being terminated according to plan B.
DCADM_STAT_SWAP	10	The system is being switched.

When the return value is negative (indicating an error):

Return value	Return value (numeric)	Explanation
DCADMER_NO_MORE_ENTRY	-1865	There is no more OpenTP1 node. The statuses of all OpenTP1 nodes have been acquired.
DCADMER_COMM	-1851	An inter-process communication error occurred.
DCADMER_PARAM	-1852	The value specified for the argument is invalid.
DCADMER_PROTO	-1854	The function dc_adm_get_nd_status_begin() was not called.
		The function dc_rpc_open() was not called.

Return value	Return value (numeric)	Explanation
DCADMER_MULTI_DEF	-1864	N is specified for multi_node_option in the system common definition.
		The TP1/Multi is not installed in the system.
		The correct version TP1/Multi is not installed in the system.
DCADMER_REMOTE	-1866	The OpenTP1 node identified by the node identifier returned to node_id cannot use the multinode facility for the following reason: • N is specified for multi_node_option in the system common definition. • The TP1/Multi is not installed in the system. • The correct version TP1/Multi is not installed in the system. The memory became insufficient. • The memory became insufficient.

dc_adm_get_nodeconf_begin - Start acquiring a node identifier

Format

- ANSI C, C++

```
#include <dcadm.h>
int dc_adm_get_nodeconf_begin (char *sub_area,
                               DCLONG *entry_count,
                               DCLONG flags)
```

- K&R C

```
#include <dcadm.h>
int dc_adm_get_nodeconf_begin (sub_area, entry_count,
                               flags)
char       *sub_area;
DCLONG     *entry_count;
DCLONG      flags;
```

Description

The function `dc_adm_get_nodeconf_begin()` starts acquiring all node identifiers in a specified multinode subarea. When this function terminates normally, it returns the number of OpenTP1 nodes.

Arguments whose value is set in the UAP

- `sub_area`

  Specify the pointer to the multinode subarea identifier or character string (`*`). Add a null character after the multinode subarea identifier. If the pointer to the character string (`*`) is specified, the function will acquire all node identifiers making up the multinode area.

  The length of the multinode subarea identifier must be equal to or less than the maximum length defined by `DCADM_SUB_AREA_NAME_SIZE`. If a longer identifier is specified, the function returns with an error.

- `entry_count`

  Specify the pointer to the area to which the number of OpenTP1 nodes will be returned. The area set here will contain the number of OpenTP1 nodes in the multinode subarea identified by `sub_area`. If the pointer to the character string (`*`) is specified for `sub_area`, the number of all OpenTP1 nodes in the multinode area will returned to the area.

- `flags`

  Specify `DCNOFLAGS`.

Return values

Return value	Return value (numeric)	Explanation
DC_OK	0	Normal termination. The area indicated by entry_count now contains the number of OpenTP1 nodes.
DCADMER_PARAM	-1852	The value specified for the argument is invalid.
DCADMER_SUBAREA_NOT_EXIST	-1860	There is no multinode subarea with the name specified for sub_area.
DCADMER_MEMORY	-1861	The memory became insufficient.
DCADMER_DEF	-1862	An incorrect value is specified in the multinode configuration definition or in the multinode physical definition.
DCADMER_PROTO	-1854	The function dc_adm_get_nodeconf_begin() was already called.
		The function dc_rpc_open() was not called.
DCADMER_MULTI_DEF	-1864	N is specified for multi_node_option in the system common definition.
		The TP1/Multi is not installed in the system.
		The correct version TP1/Multi is not installed in the system.

dc_adm_get_nodeconf_done - Terminate acquiring a node identifier

Format

■ ANSI C, C++

```
#include <dcadm.h>
int dc_adm_get_nodeconf_done (DCLONG flags)
```

■ K&R C

```
#include <dcadm.h>
int dc_adm_get_nodeconf_done (flags)
DCLONG      flags;
```

Description

The function dc_adm_get_nodeconf_done() terminates acquiring a node identifier. Call this function when the return value from the function dc_adm_get_nodeconf_begin() is DC_OK.

Arguments whose value is set in the UAP

■ flags

Specify DCNOFLAGS.

Return values

Return value	Return value (numeric)	Explanation
DC_OK	0	Normal termination.
DCADMER_PARAM	-1852	The value specified for the argument is invalid.
DCADMER_PROTO	-1854	The function dc_adm_get_nodeconf_begin() was not called.
		The function dc_rpc_open() was not called.
DCADMER_MULTI_DEF	-1864	N is specified for multi_node_option in the system common definition.
		The TP1/Multi is not installed in the system.
		The correct version TP1/Multi is not installed in the system.

dc_adm_get_nodeconf_next - Acquire a node identifier

Format

- ANSI C, C++

```
#include <dcadm.h>
int dc_adm_get_nodeconf_next (char *node_id, DCLONG flags)
```

- K&R C

```
#include <dcadm.h>
int dc_adm_get_nodeconf_next (node_id, flags)
char       *node_id;
DCLONG      flags;
```

Description

The function dc_adm_get_nodeconf_next() acquires the node identifier of one node in the multinode area containing the user server which has called this function or one node in a multinode subarea.

The data acquired by this function is data which was effective when the function dc_adm_get_nodeconf_begin() was called.

Arguments whose value is set in the UAP

- node_id

  Specify the pointer to the area which will receive the node identifier. A null character is added at the end of the node identifier. The length of the area must be equal to the length defined by DCADM_NODE_ID_SIZE.

- flags

  Specify DCNOFLAGS.

Return values

Return value	Return value (numeric)	Explanation
DC_OK	0	Normal termination.
DCADMER_NO_MORE_ENTRY	-1865	There is no more OpenTP1 node. All node identifiers have been acquired.
DCADMER_PARAM	-1852	The value specified for the argument is invalid.

Return value	Return value (numeric)	Explanation
DCADMER_PROTO	-1854	The function dc_adm_get_nodeconf_begin() was not called.
		The function dc_rpc_open() was not called.
DCADMER_MULTI_DEF	-1864	N is specified for multi_node_option in the system common definition.
		The TP1/Multi is not installed in the system.
		The correct version TP1/Multi is not installed in the system.

dc_adm_get_node_id - Acquire the node identifier of the local node

Format

- ANSI C, C++

```
#include <dcadm.h>
int dc_adm_get_node_id (char *node_id, DCLONG flags)
```

- K&R C

```
#include <dcadm.h>
int dc_adm_get_node_id (node_id, flags)
char       *node_id;
DCLONG      flags;
```

Description

The function dc_adm_get_node_id() returns the node identifier of the local OpenTP1 node specified in the system common definition to the area identified by node_id.

Arguments whose value is set in the UAP

- node_id

  Specify the pointer to the area which will receive the node identifier. A null character is added at the end of the node identifier. The length of the area must be equal to the length defined by DCADM_NODE_ID_SIZE.

- flags

  Specify DCNOFLAGS.

Return values

Return value	Return value (numeric)	Explanation
DC_OK	0	Normal termination.
DCADMER_PARAM	-1852	The value specified for the argument is invalid.
DCADMER_PROTO	-1854	The function dc_rpc_open() was not called.

dc_adm_get_sv_status - Acquire the status of a specified user server

Format

- ANSI C, C++

```
#include <dcadm.h>
int dc_adm_get_sv_status (char *node_id, char *sv_name,
                          DCLONG flags)
```

- K&R C

```
#include <dcadm.h>
int dc_adm_get_sv_status (node_id, sv_name, flags)
char       *node_id;
char       *sv_name;
DCLONG      flags;
```

Description

The function `dc_adm_get_sv_status()` acquires the status of a user server in a specified node identifier.

Arguments whose value is set in the UAP

- `node_id`

  Specify the pointer to the node identifier or the character string (`*`). Add a null character after the node identifier. If the pointer to the character string (`*`) is specified, the OpenTP1 node which called this function is assumed.

  The length of the node identifier must be equal to the length defined by `DCADM_NODE_ID_LEN`. If a node identifier with a different length is specified, the function returns with an error.

- `sv_name`

  Specify the pointer to the area containing the user server name. The length of the user server name must be equal to the length defined by `SERVER_NAME_SIZE`. If a user server name with a longer length is specified, the function returns with an error.

- `flags`

  Specify `DCNOFLAGS`.

Return values

When the return value is positive (indicating the status of the user server):

Return value	Return value (numeric)	Explanation
DCADM_STAT_TERM	8	The user server is halted or is being terminated abnormally.
DCADM_STAT_START_NORMAL	2	The user server is normally being started.
DCADM_STAT_START_RECOVER	3	The user server is being restarted.
DCADM_STAT_ONLINE	4	The user server is online.
DCADM_STAT_STOP	5	The user server is normally being terminated.

When the return value is negative (indicating an error):

Return value	Return value (numeric)	Explanation
DCADMER_PARAM	-1852	The value specified for the argument is invalid.
DCADMER_COMM	-1851	Communication with the specified OpenTP1 node is impossible for the following reason: • The OpenTP1 at the OpenTP1 node must be defined or redefined with the dcsetup command. • The value specified in the multinode physical definition is incorrect (the OpenTP1 node is not defined or the specified host name or port number is incorrect). • A communication error occurred (power is not supplied to the OpenTP1 node machine or a network error occurred).
DCADMER_MEMORY	-1861	The memory became insufficient.
DCADMER_PROTO	-1854	The function dc_rpc_open() was not called.
DCADMER_MULTI_DEF	-1864	N is specified for multi_node_option in the system common definition or an incorrect value is specified in the multinode physical definition.
		The TP1/Multi is not installed in the system.
		The correct version TP1/Multi is not installed in the system.
DCADMER_DEF	-1862	An incorrect value is specified in the multinode configuration definition.
DCADMER_NODE_NOT_EXIST	-1867	The node identified by node_id is not included in the OpenTP1 nodes.

Return value	Return value (numeric)	Explanation
DCADMER_REMOTE	-1866	The specified OpenTP1 node cannot use the multinode facility for the following reason: • N is specified for multi_node_option in the system common definition. • The TP1/Multi is not installed in the system. • The correct version TP1/Multi is not installed in the system. • The memory became insufficient.
DCADMER_SWAP	-1868	The status of the user server cannot be acquired because the system is being switched.

dc_adm_get_sv_status_begin - Start acquiring the status of a user server

Format

- ANSI C, C++

```
#include <dcadm.h>
int dc_adm_get_sv_status_begin (char *node_id,
                                DCLONG *entry_count,
                                DCLONG flags)
```

- K&R C

```
#include <dcadm.h>
int dc_adm_get_sv_status_begin (node_id, entry_count,
                                flags)
char       *node_id;
DCLONG     *entry_count;
DCLONG      flags;
```

Description

The function dc_adm_get_sv_status_begin() starts acquiring the statuses of user servers at a specified node identifier. When this function terminates normally, it returns the number of user servers whose status is to be acquired.

Arguments whose value is set in the UAP

- node_id

  Specify the pointer to the node identifier or the character string (*). Add a null character after the node identifier. If the pointer to the character string (*) is specified, the OpenTP1 node which called this function is assumed.

  The length of the node identifier must be equal to the length defined by DCADM_NODE_ID_LEN. If a node identifier with a different length is specified, the function returns with an error.

- entry_count

  Specify the pointer to the area to which the number of user servers will be returned. The area set here will contain the number of user servers at the OpenTP1 node identified by node_id.

- flags

  Specify DCNOFLAGS.

Return values

Return value	Return value (numeric)	Explanation
DC_OK	0	Normal termination. The area indicated by entry_count now contains the number of user servers.
DCADMER_PARAM	-1852	The value specified for the argument is invalid.
DCADMER_COMM	-1851	Communication with the specified OpenTP1 node is impossible for the following reason: • The OpenTP1 at the OpenTP1 node must be defined or redefined with the dcsetup command. • The value specified in the multinode physical definition is incorrect (the OpenTP1 node is not defined or the specified host name or port number is incorrect). • A communication error occurred (power is not supplied to the OpenTP1 node machine or a network error occurred).
DCADMER_MEMORY	-1861	The memory became insufficient.
DCADMER_PROTO	-1854	The function dc_adm_get_sv_status_begin() was already called.
		The function dc_rpc_open() was not called.
DCADMER_MULTI_DEF	-1864	N is specified for multi_node_option in the system common definition.
		The TP1/Multi is not installed in the system.
		The correct version TP1/Multi is not installed in the system.
DCADMER_DEF	-1862	An incorrect value is specified in the multinode configuration definition or in the multinode physical definition.
DCADMER_NODE_NOT_EXIST	-1867	The node identified by node_id is not included in the OpenTP1 nodes.
DCADMER_REMOTE	-1866	The specified OpenTP1 node cannot use the multinode facility for the following reason: • N is specified for multi_node_option in the system common definition. • The TP1/Multi is not installed in the system. • The correct version TP1/Multi is not installed in the system. • The memory became insufficient.

Return value	Return value (numeric)	Explanation
DCADMER_SWAP	-1868	The status of the user server cannot be acquired because the system is being switched.

dc_adm_get_sv_status_done - Terminate acquiring the status of a user server

Format

■ ANSI C, C++

```
#include <dcadm.h>
int dc_adm_get_sv_status_done (DCLONG flags)
```

■ K&R C

```
#include <dcadm.h>
int dc_adm_get_sv_status_done (flags)
DCLONG      flags;
```

Description

The function `dc_adm_get_sv_status_done()` terminates acquiring the status of a user server. Call this function when the return value from the function `dc_adm_get_sv_status_begin()` is `DC_OK`.

Arguments whose value is set in the UAP

■ `flags`

Specify `DCNOFLAGS`.

Return values

Return value	Return value (numeric)	Explanation
`DC_OK`	`0`	Normal termination.
`DCADMER_PARAM`	`-1852`	The value specified for the argument is invalid.
`DCADMER_PROTO`	`-1854`	The function `dc_adm_get_sv_status_begin()` was not called.
		The function `dc_rpc_open()` was not called.
`DCADMER_MULTI_DEF`	`-1864`	`N` is specified for `multi_node_option` in the system common definition.
		The TP1/Multi is not installed in the system.
		The correct version TP1/Multi is not installed in the system.

dc_adm_get_sv_status_next - Acquire the status of a user server

Format

- ANSI C, C++

```
#include <dcadm.h>
int dc_adm_get_sv_status_next (char *sv_name,
                               DCLONG flags)
```

- K&R C

```
#include <dcadm.h>
int dc_adm_get_sv_status_next (sv_name, flags)
char       *sv_name;
DCLONG      flags;
```

Description

The function `dc_adm_get_sv_status_next()` acquires the statuses of user servers at a specified OpenTP1 node.

The data acquired by this function is data which was effective when the function `dc_adm_get_sv_status_begin()` was called.

Arguments whose value is set in the UAP

- `sv_name`

  Specify the pointer to the area which will receive the user server name. The length of the area must be equal to the length defined by `SERVER_NAME_SIZE`.

- `flags`

  Specify `DCNOFLAGS`.

Return values

When the return value is positive (indicating the status of the user server):

Return value	Return value (numeric)	Explanation
`DCADM_STAT_TERM`	8	The user server is halted or is being terminated abnormally.
`DCADM_STAT_START_NORMAL`	2	The user server is normally being started.
`DCADM_STAT_START_RECOVER`	3	The user server is being restarted.
`DCADM_STAT_ONLINE`	4	The user server is online.

Return value	Return value (numeric)	Explanation
DCADM_STAT_STOP	5	The user server is normally being terminated.

When the return value is negative (indicating an error):

Return value	Return value (numeric)	Explanation
DCADMER_NO_MORE_ENTRY	-1865	There is no more user server. The statuses of all user servers have been acquired.
DCADMER_PARAM	-1852	The value specified for the argument is invalid.
DCADMER_PROTO	-1854	The function dc_adm_get_sv_status_begin() was not called.
		The function dc_rpc_open() was not called.
DCADMER_MULTI_DEF	-1864	N is specified for multi_node_option in the system common definition.
		The TP1/Multi is not installed in the system.
		The correct version TP1/Multi is not installed in the system.

DAM file service (dc_dam_~)

This section gives the syntax and other information of the following functions which are used for DAM file service:

Functions that can only be used in an online environment

- `dc_dam_close` - Close a logical file
- `dc_dam_end` - Terminate using an unrecoverable DAM file
- `dc_dam_hold` - Shut down a logical file
- `dc_dam_open` - Open a logical file
- `dc_dam_read` - Input a logical file block
- `dc_dam_release` - Release a logical file from the shutdown state
- `dc_dam_rewrite` - Update a logical file block
- `dc_dam_start` - Start using an unrecoverable DAM file
- `dc_dam_status` - Reference the status of a logical file
- `dc_dam_write` - Output a logical file block

Functions that can only be used in an offline environment

- `dc_dam_bseek` - Seek a physical file block
- `dc_dam_create` - Allocate a physical file
- `dc_dam_dget` - Input directly a physical file block
- `dc_dam_dput` - Output directly a physical file block
- `dc_dam_get` - Input a physical file block
- `dc_dam_iclose` - Close a physical file
- `dc_dam_iopen` - Open a physical file
- `dc_dam_put` - Output a physical file block

The functions for DAM file service (`dc_dam_~`) can be used only in UAPs of TP1/Server Base. They cannot be used in UAPs of TP1/LiNK.

dc_dam_bseek - Seek a physical file block

Format

- ANSI C, C++

```
#include <dcdami.h>
int dc_dam_bseek (int fno, int blkno, DCLONG flags)
```

- K&R C

```
#include <dcdami.h>
int dc_dam_bseek (fno, blkno, flags)
int        fno;
int        blkno;
DCLONG     flags;
```

Description

The function `dc_dam_bseek()` specifies the relative block number of a physical file to position the file at the corresponding block. Call this function after the function `dc_dam_iopen()` that requests re-creation output.

When the corresponding relative block number is in the file, the relative block number is returned without modification.

When seeking a physical file block, specify the file descriptor which is the return value of the function `dc_dam_iopen()`.

Arguments whose values are set in the UAP

- `fno`

  Specify the file descriptor of the file containing a block to be located.

- `blkno`

  Specify the relative block number to be located.

- `flags`

  Specify `DCNOFLAGS`.

Return values

Return value	Return value (numeric)	Explanation
0 or positive integer		The value `0` or a positive integer indicates a relative block number.

Return value	Return value (numeric)	Explanation
DCDAMER_BADF	-1603	The file descriptor specified for fno is not the one which was acquired by opening the file normally.
		The DAM file is not open.
DCDAMER_SEQER	-1605	The call sequence of functions which access the DAM file is invalid.
DCDAMER_BNOER	-1606	The relative block number is invalid.
DCDAMER_PARAM_FLAGS	-1611	The value specified for flags is invalid.
DCDAMER_IOER	-1620	An output error occurred.

dc_dam_close - Close a logical file

Format

- ANSI C, C++

```
#include <dcdam.h>
int dc_dam_close (int damfd, DCLONG flags)
```

- K&R C

```
#include <dcdam.h>
int dc_dam_close (damfd, flags)
int      damfd;
DCLONG   flags;
```

Description

The function `dc_dam_close()` closes logical files.

- For recoverable DAM files

  If a logical file opened within the transaction is not closed before the transaction terminates, the DAM service closes it at the synchronization point processing. However, the DAM service does not close a logical file opened outside the transaction (before the function `dc_trn_begin()` is called) or an unrecoverable DAM file.

  If a logical file is opened before the transaction is started, it must be closed before the UAP processing is terminated.

- For unrecoverable DAM files

  Since a logical file is not synchronized with the transaction, the function `dc_dam_close()` can arbitrarily be called when a logical file is closed. However, opened logical files must be closed with the function `dc_dam_close()` before the function `dc_dam_end()` is called.

When closing a logical file, specify the file descriptor which is the return value of the function `dc_dam_open()`.

Arguments whose values are set in the UAP

- `damfd`

  Specify the file descriptor of the file to be closed.

- `flags`

  Specify `DCNOFLAGS`.

Return values

Return value	Return value (numeric)	Explanation
DC_OK	0	The logical file was closed normally.
DCDAMER_PROTO	-1600	The function dc_rpc_open() is not called.
		A DAM file opened outside the transaction is closed within the transaction. (This value is returned only when a recoverable DAM file is accessed.)
		N is specified for atomic_update in the user service definition. (This value is returned only when a recoverable DAM file is accessed.)
		The function dc_dam_start() is not called. (This value is returned only when an unrecoverable DAM file is accessed.)
		The UAP is incorrectly linked as follows: • The library (-ltdam) to be used for access to a TAM file using a DAM service function is linked incorrectly. • The definition of the resource manager for transaction control object files is incorrect.
DCDAMER_BADF	-1603	The file descriptor specified for damfd is not the one which was acquired by opening the file normally, or the file of the file descriptor is not open.
DCDAMER_PARAM_FLAGS	-1611	The value specified for flags is invalid.

dc_dam_create - Allocate a physical file

Format

- ANSI C, C++

```
#include <dcdami.h>
int dc_dam_create (char *fname, int blksize, int blknum,
                   int pnum, DCLONG flags)
```

- K&R C

```
#include <dcdami.h>
int dc_dam_create (fname, blksize, blknum, pnum, flags)
char     *fname;
int      blksize;
int      blknum;
int      pnum;
DCLONG   flags;
```

Description

The function `dc_dam_create()` allocates a physical file to the OpenTP1 file system.

The size of a physical file is (block length + 8) x (number of blocks + 1).

Calling the function `dc_dam_iopen()` is unnecessary after the function `dc_dam_create()` is called.

The following functions cannot be called after the function `dc_dam_create()` is called:

- `dc_dam_get()`
- `dc_dam_bseek()`
- `dc_dam_dget()`
- `dc_dam_dput()`

The size of an output buffer is (block length + 8) x (number of blocks collectively processed).

Arguments whose values are set in the UAP

- `fname`

  Specify the name of a physical file to be created in the OpenTP1 file system, with a path name. The path name must be within (special file name + 14) bytes.

- `blksize`

  Specify the length of a physical file block.

- `blknum`

  Specify the number of physical file blocks.

- `pnum`

  Specify the number of blocks collectively processed which is used as an input/output unit.

- `flags`

  Specify the access permissions of the owner, the owner group, and another UAP. The access permissions must be specified with the values shown below or the bit strings shown in parentheses.

  `DCDAM_READ_OWNER` (00400): The read permission of the owner is specified.

  `DCDAM_WRITE_OWNER` (00200): The write permission of the owner is specified.

  `DCDAM_READ_GROUP` (00040): The read permission of the group owner is specified.

  `DCDAM_WRITE_GROUP` (00020): The write permission of the group owner is specified.

  `DCDAM_READ_OTHERS` (00004): The read permission of another UAP is specified.

  `DCDAM_WRITE_OTHERS` (00002): The write permission of another UAP is specified.

  The following values are assumed when `DCNOFLAGS` is specified:

  `DCDAM_READ_OWNER` (00400)

  `DCDAM_WRITE_OWNER` (00200)

  `DCDAM_READ_GROUP` (00040)

  `DCDAM_READ_OTHERS` (00004)

Return values

Return value	Return value (numeric)	Explanation
0 or positive integer		0 or a positive integer indicates the file descriptor.
`DCDAMER_NOMEM`	`-1607`	The memory became insufficient.
`DCDAMER_OPENED`	`-1608`	The specified physical file is opened.
`DCDAMER_PARAM_FLAGS`	`-1611`	The value specified for `flags` is invalid.
`DCDAMER_FILEER`	`-1614`	The physical file name is invalid.

Return value	Return value (numeric)	Explanation
DCDAMER_PNUMER	-1615	The value specified for the number of blocks collectively processed is invalid.
DCDAMER_EXIST	-1617	A physical file having the same name has been already allocated.
DCDAMER_VERSION	-1618	The OpenTP1 file system versions used for creation and allocation do not match each other.
DCDAMER_IOER	-1620	An input/output error occurred.
DCDAMER_ACCESS	-1628	The UAP that called the function dc_dam_create() does not have the access permission for special files.
		A DAM file to be allocated is protected with the security facility. The UAP that called the function dc_dam_create() has no access permission.
DCDAMER_LBLNER	-1630	The value specified for the block length is not suitable.
DCDAMER_LBNOER	-1631	The value specified for the number of blocks is not suitable.
DCDAMER_LFNMER	-1632	The physical file is not a character special file, or the device corresponding to the special file does not exist.
DCDAMER_LNOINT	-1633	The specified OpenTP1 file has not been initialized as an OpenTP1 file system.
DCDAMER_LFFOVF	-1634	When the OpenTP1 file was initialized as an OpenTP1 file system, an attempt was made to allocate more OpenTP1 files (physical files) than specified.
DCDAMER_LFNOVF	-1635	The specified value exceeds the maximum number of files which can be opened in the process being executed.
DCDAMER_USED	-1636	The physical file specified for fname is being used in online mode, or it is being used by another process.
DCDAMER_SPACE	-1640	The OpenTP1 file system does not have a free area large enough to allocate physical files.
DCDAMER_NO_ACL	-1646	A DAM file to be allocated is protected with the security facility. There is no ACL for the corresponding file.

dc_dam_dget - Input directly a physical file block

Format

- ANSI C, C++

```
#include <dcdami.h>
int dc_dam_dget (int fno, char *datadr, int datalen,
                 int blkno, DCLONG flags)
```

- K&R C

```
#include <dcdami.h>
int dc_dam_dget (fno, datadr, datalen, blkno, flags)
int        fno;
char       *datadr;
int        datalen;
int        blkno;
DCLONG     flags;
```

Description

The function `dc_dam_dget()` inputs a block corresponding to a specified relative block number. Call this function after the function `dc_dam_iopen()` that requests re-creation output.

If the value specified for the block length is less than the value specified for the buffer length, the length of the input block is returned. If the value specified for the block length is greater than the value specified for the buffer length, an error is returned.

When directly inputting a physical file block, specify the file descriptor which is the return value of the function `dc_dam_iopen()`.

Arguments whose values are set in the UAP

- `fno`

  Specify the file descriptor of the file containing a block to be input directly.

- `datadr`

  Specify the address of the input buffer.

- `datalen`

  Specify the length of the input buffer.

- `blkno`

  Specify the relative block number of the input block.

- `flags`

  Specify `DCNOFLAGS`.

Return values

Return value	Return value (numeric)	Explanation
Positive integer		A positive integer indicates the length of the input block.
`DCDAMER_BADF`	`-1603`	The file descriptor specified for `fno` is not the one which was acquired by opening the file normally.
		The DAM file is not open.
`DCDAMER_BUFER`	`-1604`	The value specified for the input data length is less than the value specified for the block length.
`DCDAMER_SEQER`	`-1605`	The call sequence of functions which access the DAM file is invalid.
`DCDAMER_BNOER`	`-1606`	The relative block number is invalid.
`DCDAMER_PARAM_FLAGS`	`-1611`	The value specified for `flags` is invalid.
`DCDAMER_IOER`	`-1620`	An input error occurred.
`DCDAMER_ACCESS`	`-1628`	A DAM file to be accessed is protected with the security facility. The UAP that called the function `dc_dam_dget()` has no access permission.
`DCDAMER_NO_ACL`	`-1646`	A DAM file to be accessed is protected with the security facility. There is no ACL for the corresponding file.

dc_dam_dput - Output directly a physical file block

Format

- ANSI C, C++

```
#include <dcdami.h>
int  dc_dam_dput (int fno, char *datadr, int datalen,
                  int blkno, DCLONG flags)
```

- K&R C

```
#include <dcdami.h>
int  dc_dam_dput (fno, datadr, datalen, blkno, flags)
int        fno;
char       *datadr;
int        datalen;
int        blkno;
DCLONG     flags;
```

Description

The function `dc_dam_dput()` outputs a block corresponding to a specified relative block number. Call this function after the function `dc_dam_iopen()` that requests re-creation output.

If the value specified for the output data length is less than the value specified for the block length, a block is output and the remaining area is padded with null characters. If the value specified for the output data length is greater than the value specified for the block length, an error is returned.

When directly outputting a physical file block, specify the file descriptor which is the return value of the function `dc_dam_iopen()`.

Arguments whose values are set in the UAP

- `fno`

  Specify the file descriptor of the file to which a block is output directly.

- `datadr`

  Specify the address of the output data.

- `datalen`

  Specify the length of the output data.

- `blkno`

  Specify the relative block number of the output destination block.

- `flags`

  Specify `DCNOFLAGS`.

Return values

Return value	Return value (numeric)	Explanation
Positive integer		A positive integer indicates the length of the output block.
`DCDAMER_BADF`	`-1603`	The file descriptor specified for `fno` is not the one which was acquired by opening the file normally.
		The DAM file is not open.
`DCDAMER_BUFER`	`-1604`	The value specified for the output data length is less than the value specified for the block length.
`DCDAMER_SEQER`	`-1605`	The call sequence of functions which access the DAM file is invalid.
`DCDAMER_BNOER`	`-1606`	The relative block number is invalid.
`DCDAMER_PARAM_FLAGS`	`-1611`	The value specified for `flags` is invalid.
`DCDAMER_IOER`	`-1620`	An output error occurred.
`DCDAMER_ACCESS`	`-1628`	A DAM file to be accessed is protected with the security facility. The UAP that called the function `dc_dam_dput()` has no access permission.
`DCDAMER_NO_ACL`	`-1646`	A DAM file to be accessed is protected with the security facility. There is no ACL for the corresponding file.

dc_dam_end - Terminate using an unrecoverable DAM file

Format

- ANSI C, C++

```
#include <dcdam.h>
int  dc_dam_end (DCLONG flags)
```

- K&R C

```
#include <dcdam.h>
int  dc_dam_end (flags)
DCLONG     flags;
```

Description

The function `dc_dam_end()` terminates using an unrecoverable DAM file.

When the function `dc_dam_start()` is called, call the function `dc_dam_end()` before terminating the processing. If the function `dc_dam_end()` is not called, a resource used to access an unrecoverable DAM file is not released until the UAP terminates.

Argument whose value is set in the UAP

- `flags`

  Specify `DCNOFLAGS`.

Return values

Return value	Return value (numeric)	Explanation
`DC_OK`	`0`	Normal termination. Using an unrecoverable DAM file is terminated.
`DCDAMER_PROTO`	`-1600`	The function `dc_rpc_open()` is not called.
`DCDAMER_SEQER`	`-1605`	The function `dc_dam_start()` is not called.
`DCDAMER_PARAM_FLAGS`	`-1611`	The value specified for `flags` is invalid.

dc_dam_get - Input a physical file block

Format

- ANSI C, C++

```
#include <dcdami.h>
int  dc_dam_get (int fno, char *datadr, int datalen,
                 DCLONG flags)
```

- K&R C

```
#include <dcdami.h>
int  dc_dam_get (fno, datadr, datalen, flags)
int      fno;
char     *datadr;
int      datalen;
DCLONG   flags;
```

Description

The function `dc_dam_get()` sequentially inputs data in blocks from a physical file of the OpenTP1 file system. Call the function `dc_dam_get()` after the function `dc_dam_iopen()`.

If the value specified for the block length is smaller than the value specified for the buffer length, the length of the input block is returned. If the value specified for the block length is greater than the value specified for buffer length, an error is returned.

When inputting a physical file block, specify the file descriptor which is the return value of the function `dc_dam_iopen()`.

Arguments whose values are set in the UAP

- `fno`

  Specify the file descriptor of the file containing a block to be input.

- `datadr`

  Specify the address of the input buffer.

- `datalen`

  Specify the length of the input buffer. You can specify a value in the range from 504 to 2147483647.

- `flags`

  Specify `DCNOFLAGS`.

Return values

Return value	Return value (numeric)	Explanation
Positive integer		A positive integer indicates the length of the input block.
DCDAMER_BADF	-1603	The file descriptor specified for fno is not the one which was acquired by opening the file normally, or the file is not open.
DCDAMER_BUFER	-1604	The value specified for the block length is greater than the value specified for the buffer length.
		The value specified for the input buffer length is outside the range of values that can be specified.
DCDAMER_SEQER	-1605	The call sequence of functions which access the DAM file is invalid.
DCDAMER_PARAM_FLAGS	-1611	The value specified for flags is invalid.
DCDAMER_IOER	-1620	An input error occurred.
DCDAMER_ACCESS	-1628	A DAM file to be accessed is protected with the security facility. The UAP that called the function dc_dam_get() has no access permission.
DCDAMER_EOF	-1637	The file end was reached.
DCDAMER_NO_ACL	-1646	A DAM file to be accessed is protected with the security facility. There is no ACL for the corresponding file.

dc_dam_hold - Shut down a logical file

Format

- ANSI C, C++

```
#include <dcdam.h>
int  dc_dam_hold (char *lfname, DCLONG flags)
```

- K&R C

```
#include <dcdam.h>
int  dc_dam_hold (lfname, flags)
char     *lfname;
DCLONG   flags;
```

Description

The function `dc_dam_hold()` shuts down a logical file. After the function `dc_dam_hold()` is executed, a logical shutdown error is always returned if another UAP calls an access request for the logical file.

- For recoverable DAM files

  If the logical file specified here is under synchronization point processing in another transaction processing when the function `dc_dam_hold()` is called, the logical file is closed after the synchronization point processing terminates. Even if the synchronization point processing is not completed, the function `dc_dam_hold()` returns to the accessed UAP.

Arguments whose values are set in the UAP

- `lfname`

  Within 1 to 8 bytes, specify the name of a logical file to be shut down.

- `flags`

  Specify `DCNOFLAGS`.

Return values

Return value	Return value (numeric)	Explanation
`DC_OK`	`0`	The logical file specified for `lfname` was shut down normally.
`DCDAMER_PROTO`	`-1600`	The function `dc_rpc_open()` is not called.

Return value	Return value (numeric)	Explanation
		N is specified for atomic_update in the user service definition. (This value is returned only when a recoverable DAM file is accessed.)
		The function dc_dam_start() is not called. (This value is returned only when an unrecoverable DAM file is accessed.)
		The UAP is incorrectly linked as follows: • The library (-ltdam) to be used for access to a TAM file using a DAM service function is linked incorrectly. • The definition of the resource manager for transaction control object files is incorrect.
DCDAMER_UNDEF	-1601	The specified logical file name has not been defined.
DCDAMER_NOMEM	-1607	The memory became insufficient.
DCDAMER_PARAM_LFNAME	-1610	The logical file name specified for lfname is invalid.
DCDAMER_PARAM_FLAGS	-1611	The value specified for flags is invalid.
DCDAMER_VERSION	-1618	The version of the DAM library linked to the UAP does not allow the UAP to operate with the current DAM service.
DCDAMER_LHOLDED	-1625	The logical file name specified for lfname is in logical shutdown state.
DCDAMER_OHOLDED	-1626	The logical file name specified for lfname is in shutdown state due to an error.
DCDAMER_ACCESS	-1628	A DAM file to be shut down is protected with the security facility. The UAP that called the function dc_dam_hold() has no access permission.
DCDAMER_NO_ACL	-1646	A DAM file to be shut down is protected with the security facility. There is no ACL for the corresponding file.

dc_dam_iclose - Close a physical file

Format

■ ANSI C, C++

```
#include <dcdami.h>
int  dc_dam_iclose (int fno, DCLONG flags)
```

■ K&R C

```
#include <dcdami.h>
int  dc_dam_iclose (fno, flags)
int      fno;
DCLONG   flags;
```

Description

The function `dc_dam_iclose()` closes a physical file created in the OpenTP1 file system.

If a file is not filled with data, the remaining part up to the end of the file is padded with blocks of null characters only in the following cases:

- The value specified for flags of the function `dc_dam_iopen()` indicates a creation output request (`DCDAM_INITIALIZE`).
- The function `dc_dam_create()` has been called.

When closing a physical file, specify the file descriptor which is the return value of the function `dc_dam_create()` or `dc_dam_iopen()`.

Arguments whose values are set in the UAP

■ `fno`

Specify the file descriptor of the file to be closed.

■ `flags`

Specify `DCNOFLAGS`.

Return values

Return value	Return value (numeric)	Explanation
`DC_OK`	`0`	The physical file was closed normally.
`DCDAMER_BADF`	`-1603`	The file descriptor specified for `fno` is not the one which was acquired by opening the file normally.

Return value	Return value (numeric)	Explanation
		The specified file is not open.
DCDAMER_PARAM_FLAGS	-1611	The value specified for `flags` is invalid.
DCDAMER_IOER	-1620	An output error occurred.

dc_dam_iopen - Open a physical file

Format

- ANSI C, C++

```
#include <dcdami.h>
int dc_dam_iopen (char *fname, int pnum, DCLONG flags)
```

- K&R C

```
#include <dcdami.h>
int dc_dam_iopen (fname, pnum, flags)
char     *fname;
int      pnum;
DCLONG   flags;
```

Description

The function `dc_dam_iopen()` opens a physical file created in the OpenTP1 file system. However, this function cannot open a physical file being used in online mode.

Arguments whose values are set in the UAP

- `fname`

  Specify the name of a physical file to be opened with a path name within (special file name + 14 bytes).

- `pnum`

  Specify the number of blocks collectively processed which is used as an input/output unit.

- `flags`

  Specify the type of request (creation output request or re-creation (overwrite) output request). The value specified here determines whether to pad the remaining area with blocks of null characters when the file is closed. The value set here will come into effect when the call of the function `dc_dam_iclose()` subsequent to the function `dc_dam_put()` brings about normal termination. Even though the function `dc_dam_put()` is called, the remaining area will not be padded with blocks of null characters provided that UAP processing is terminated without the call of the function `dc_dam_iclose()`.

  `DCDAM_INITIALIZE`

  The creation output request type is specified. (The remaining area is padded with blocks of null characters.)

DCDAM_OVERWRITE

The re-creation output request type is specified. (The remaining area is not padded with blocks of null characters.)

When DCNOFLAGS is specified, DCDAM_OVERWRITE is assumed to be specified.

Return values

Return value	Return value (numeric)	Explanation
0 or positive integer		0 or a positive integer indicates the file descriptor.
DCDAMER_NOMEM	-1607	The memory became insufficient.
DCDAMER_OPENED	-1608	The physical file specified for fname is open.
DCDAMER_PARAM_FLAGS	-1611	The value specified for flags is invalid.
DCDAMER_FILEER	-1614	The physical file name specified for fname is invalid.
DCDAMER_PNUMER	-1615	The value specified for the number of blocks collectively processed is invalid.
DCDAMER_NODAM	-1616	The physical file specified for fname is not a DAM file.
DCDAMER_VERSION	-1618	The OpenTP1 file system versions used for creation and allocation do not match each other.
DCDAMER_NOEXIST	-1619	The physical file specified for fname does not exist.
DCDAMER_IOER	-1620	An input/output error occurred.
DCDAMER_ACCESS	-1628	The UAP that called the function dc_dam_iopen() does not have the access permission for special files.
DCDAMER_LFNMER	-1632	The physical file is not a character special file, or the device corresponding to the special file does not exist.
DCDAMER_LNOINT	-1633	The physical file specified for fname has not been initialized as an OpenTP1 file system.
DCDAMER_LFNOVF	-1635	The specified value exceeds the maximum number of files which can be opened for the process.
DCDAMER_USED	-1636	The physical file specified for fname is being used in online mode, or it is being used by another process.
DCDAMER_ACCESSF	-1638	The access permission for physical files has not been granted.
DCDAMER_CRUSH	-1639	Physical file damage was detected.

dc_dam_open - Open a logical file

Format

- ANSI C, C++

```
#include <dcdam.h>
int dc_dam_open (char *lfname, DCLONG flags)
```

- K&R C

```
#include <dcdam.h>
int dc_dam_open (lfname, flags)
char     *lfname;
DCLONG   flags;
```

Description

The function `dc_dam_open()` opens a logical file.

- For recoverable DAM files

  Whether to apply file-based or block-based lock is specified for the logical file. File-based lock can be applied when:

  - The logical file is opened within the transaction range under the condition that lock control for individual transaction branches is specified.

  In the following conditions, file-based lock cannot be applied. Use block-based lock:

- The logical file is opened outside the transaction range.
- Lock control for individual global transactions is specified.

  If a logical file is closed by the function `dc_dam_close()` and is again opened in the same transaction branch, the status before the function `dc_dam_close()` is called is inherited.

- For unrecoverable DAM files

  Since the transaction is not synchronized, no lock is needed.

Arguments whose values are set in the UAP

- `lfname`

  Within 1 to 8 bytes, specify the name of a logical file to be opened.

- `flags`

  Specify the following items in the format below:

- File-based lock or blocks
- Whether the function is to wait for the resource to be released from lock if a lock error occurs.

```
{DCDAM_FILE_EXCLUSIVE|DCDAM_BLOCK_EXCLUSIVE
                              [|DCDAM_WAIT|DCDAM_NOWAIT|]}
```

- Flag 1

  Specify files-based lock or blocks.

  DCDAM_FILE_EXCLUSIVE: Files-based lock

  DCDAM_BLOCK_EXCLUSIVE: Blocks-based lock

- Flag 2

  Specify whether the function is to wait for the resource to be released from lock if an lock error occurs in the function,

  DCDAM_WAIT: The function waits for the resource to be released from lock.

  DCDAM_NOWAIT: The function does not wait for the resource to be released from lock, and returns with an error.

  The default is DCDAM_NOWAIT.

Setting flags

The value specified for flags depends on whether the DAM file is recoverable.

- For recoverable DAM files

  DCDAM_WAIT (flag 2) is specified if a lock error occurs in the function dc_dam_read() or dc_dam_write(). It is not specified if a lock error occurs in the function dc_dam_open(). If a lock error occurs in the function dc_dam_open(), the function unconditionally returns with the error value DCDAMER_EXCER.

  The table below shows the correspondence between the value specified for flags and the type of lock when a recoverable DAM file is accessed.

Flag 1	Flag 2#	Lock Specified for Flags
FILE_EXCLUSIVE	--	Files-based lock
BLOCK_EXCLUSIVE	WAIT	Blocks-based lock, and waiting for release from lock if a lock error occurs
	NOWAIT	Blocks-based lock, and error return if a lock error occurs

Legend:

--: Cannot be specified.

#: The default is NOWAIT.

- For unrecoverable DAM files

  DCDAM_WAIT (flag 2) is specified if a lock error occurs. If a lock error occurs in the function dc_dam_open(), dc_dam_read(), or dc_dam_write, whether to wait for lock to be released is determined according to the value specified for flag 2. When DCDAM_NOWAIT is specified for flag 2 or omitted and if a lock error occurs, the function returns with the error value DCDAMER_EXCER.

  The table below shows the correspondence between the value specified for flags and the type of lock when an unrecoverable DAM file is accessed.

Flag 1	Flag 2#	Lock Specified for Flags
FILE_EXCLUSIVE	WAIT	Files-based lock, and waiting for release from lock if a lock error occurs
	NOWAIT	Files-based lock, and error return if a lock error occurs
BLOCK_EXCLUSIVE	WAIT	Blocks-based lock, and waiting for release from lock if a lock error occurs
	NOWAIT	Blocks-based lock, and error return if a lock error occurs

#: The default is NOWAIT.

When files-based lock is specified for flag 1, no lock error occurs in the function dc_dam_read() or dc_dam_write() because all files are locked regardless of recoverable or unrecoverable files. Therefore, whether to wait for release from lock cannot be specified. The lock release wait type specified for the argument of the function dc_dam_read() or dc_dam_write() is ignored.

Return values

Return value	Return value (numeric)	Explanation
0 or positive integer		0 or a positive integer indicates the file descriptor.
DCDAMER_PROTO	-1600	The function dc_rpc_open() is not called.
		N is specified for atomic_update in the user service definition. (This value is returned only when a recoverable DAM file is accessed.)

Return value	Return value (numeric)	Explanation
		The dc_dam_start() function is not called when N is specified for the atomic_update operand in the user service definition. (This value is returned only when an unrecoverable DAM file is accessed.)
		The UAP is incorrectly linked as follows: • The library (-ltdam) to be used for access to a TAM file using a DAM service function is linked incorrectly. • The definition of the resource manager for transaction control object files is incorrect.
		DAM file lock is specified from outside the transaction range. (This value is returned only when a recoverable DAM file is accessed.)
		File lock is specified for the DAM file in lock control for each global transaction. (This value is returned only when a recoverable DAM file is accessed.)
DCDAMER_UNDEF	-1601	The logical file name specified for lfname has not been defined.
DCDAMER_EXCER	-1602	A lock error occurred.
DCDAMER_SEQER	-1605	The dc_dam_start() function is not called when Y is specified for the atomic_update operand in the user service definition. (This value is returned only when an unrecoverable DAM file is accessed.)
DCDAMER_NOMEM	-1607	The memory became insufficient.
DCDAMER_OPENED	-1608	The logical file specified for lfname is open.
DCDAMER_PARAM_LFNAME	-1610	The value specified for the logical file name is invalid.
DCDAMER_PARAM_FLAGS	-1611	The value specified for flags is invalid.
DCDAMER_LHOLD	-1621	The file specified for lfname is in logical shutdown state.
DCDAMER_OHOLD	-1622	The file specified for lfname is in shutdown state due to an error.
DCDAMER_OPENNUM	-1627	The number of open character special files exceeds the specified limit.
DCDAMER_ACCESS	-1628	The access permission for character special files has not been granted.

Return value	Return value (numeric)	Explanation
DCDAMER_TMERR	-1629	An error occurred in the transaction service. (This value is returned only when a recoverable DAM file is accessed.)
DCDAMER_DLOCK	-1642	A deadlock occurred. (This value is returned only when an unrecoverable DAM file is accessed.)
DCDAMER_TIMOUT	-1643	The resource could not be acquired because a timeout occurred (the wait time specified in the lock service definition was exceeded). (This value is returned only when an unrecoverable DAM file is accessed.)
DCDAMER_LCKOV	-1645	The number of lock requests exceeds the specified maximum number of concurrent lock requests.
DCDAMER_NO_ACL	-1646	A DAM file to be opened is protected with the security facility. There is no ACL for the corresponding file.

dc_dam_put - Output a physical file block

Format

- ANSI C, C++

```
#include <dcdami.h>
int dc_dam_put (int fno, char *datadr, int datalen,
                DCLONG flags)
```

- K&R C

```
#include <dcdami.h>
int dc_dam_put (fno, datadr, datalen, flags)
int      fno;
char     *datadr;
int      datalen;
DCLONG   flags;
```

Description

The function `dc_dam_put()` sequentially outputs data in blocks to a physical file created in the OpenTP1 file system. If the value specified for the data length is smaller than the value specified for the block length, the remaining part following the data is padded with null characters. If the value specified for the data length is greater than the value specified for the block length, an error is returned.

When outputting a physical file block, specify the file descriptor which is the return value of the function `dc_dam_create()` or `dc_dam_iopen()`.

Argument whose values are set in the UAP

- `fno`

  Specify the file descriptor of the file to which a block is output.

- `datadr`

  Specify the address of the data to be output.

- `datalen`

  Specify the length of the data to be output. You can specify a value in the range from 504 to 2147483647.

- `flags`

  Specify `DCNOFLAGS`.

Return values

Return value	Return value (numeric)	Explanation
Positive integer		A positive integer indicates the length of the data to be output (the value specified for `datalen`).
`DCDAMER_BADF`	`-1603`	The file descriptor specified for `fno` is not the one which was acquired by opening the file normally, or the specified file is not open.
`DCDAMER_BUFER`	`-1604`	The value specified for the data length is greater than the value specified for the block length.
		The value specified for the output data length is outside the range of values that can be specified.
`DCDAMER_SEQER`	`-1605`	The call sequence of functions which access the DAM file is invalid.
`DCDAMER_PARAM_FLAGS`	`-1611`	The value specified for `flags` is invalid.
`DCDAMER_IOER`	`-1620`	An output error occurred.
`DCDAMER_ACCESS`	`-1628`	A DAM file to be accessed is protected with the security facility. The UAP that called the function `dc_dam_put()` has no access permission.
`DCDAMER_EOF`	`-1637`	The end of the file is reached.
`DCDAMER_NO_ACL`	`-1646`	A DAM file to be accessed is protected with the security facility. There is no ACL for the corresponding file.

dc_dam_read - Input a logical file block

Format

- ANSI C, C++

```
#include <dcdam.h>
int  dc_dam_read (int damfd, struct DC_DAMKEY *keyptr,
                  int keyno, char *bufadr, int bufsize,
                  DCLONG flags)
```

- K&R C

```
#include <dcdam.h>
int  dc_dam_read (damfd, keyptr, keyno, bufadr, bufsize,
                  flags)
int      damfd;
struct DC_DAMKEY *keyptr;
int      keyno;
char     *bufadr;
int      bufsize;
DCLONG   flags;
```

Description

The function `dc_dam_read()` inputs a block (which is in the specified range) for reference or update processing from the specified logical file

- For recoverable DAM files

  Lock is enabled in units (files or blocks) as specified when the logical file was opened. The function `dc_dam_read()` can be called from a process out of the transaction range. In this case, however, the function can be used only for reference and lock cannot be specified.

  When multiple blocks are specified at a time, an error returned if even one of the blocks causes an error. In this case, the blocks are not input to the input buffer. All the blocks for which an input request was made are released lock at this time.

  Lock which is enabled for a block input for reference processing is released in the following case:

  After the block is input for reference processing, an input request for update processing is made for the same block. Then, an input error occurs during the update processing.

  Even if block update during a transaction is specified (`dam_update_block_over=flush` in the DAM service definition), an error is returned with `DCDAMER_JNLOV` in the following case:

- DAM file blocks are not updated in one transaction branch (`dc_dam_rewrite()`). The function `dc_dam_read()` (block input for update processing) is called. Eventually, the number of blocks exceeds the maximum number of blocks collectively updated (the value specified for `dam_update_block` of the DAM service definition).

When inputting a block of a recoverable DAM file, call the function `dc_dam_read()` from the transaction range.

- For unrecoverable DAM files

  There is no limit on the condition to call the function `dc_dam_read()` when a block of an unrecoverable DAM file is input.

  For an unrecoverable DAM file, if the function `dc_dam_read()` for update is called more times than specified in `dam_update_block` in the DAM service definition, the function returns with the error value `DCDAMER_ACSOV`.

When inputting a logical file block, specify the file descriptor which is the return value of the function `dc_dam_open()`.

Arguments whose values are set in the UAP

- `damfd`

  Specify the file descriptor of the file containing a block to be input.

- `keyptr`

  Specify the address of the structure (DAM key) that indicates the block reference/ update range. For the structure, specify the block range with the first relative block number and the last relative block number. The structure format is as shown below.

```
struct DC_DAMKEY {
                  int fstblkno;
                  int endblkno;
                 };
```

  - `fstblkno`

    Specify the first relative block number of the block to be referenced or updated.

  - `endblkno`

    Specify the last relative block number of the block to be referenced or updated. If 0 is specified, only the block of the relative block number specified for `fstblkno` is input.

- `keyno`

  Specify the number of structures (number of structure arrays) to be set for `keyptr`.

- bufadr

  Specify the address of the input buffer.

- bufsize

  Specify the length of the input buffer. The length must be equal to or greater than (input block length x number of blocks). You can specify a value in the range from 504 to 2147483647.

- flags

  Specify the type of request (reference request or update request) in the following format:

```
{DCDAM_REFERENCE|DCDAM_MODIFY}
[|{DCDAM_EXCLUSIVE|DCDAM_NOEXCLUSIVE}]
[|{DCDAM_WAIT|DCDAM_NOWAIT}]
```

  - Flag 1

    Specify the purpose (reference or update) of the input request given by the function dc_dam_read():

    DCDAM_REFERENCE: Input request for reference

    DCDAM_MODIFY: Input request for update

  - Flag 2

    Specify whether to apply lock if the input request is for reference. If DCDAM_EXCLUSIVE is specified, lock will remain until processing reaches the synchronization point.

    To access a recoverable DAM file for reference from outside the transaction, lock cannot be specified.

    If flag 2 is omitted, DCDAM_NOEXCLUSIVE is assumed.

    If the function dc_dam_read() is called without lock application, the block could be updated by another UAP during the processing of the function dc_dam_read(). In this case, the details input to the block by the function dc_dam_read() depend on the update processing status on the other UAP. Therefore, to reference the latest block contents, be sure to specify DCDAM_EXCLUSIVE.

    If the input request is for update, flag 2 cannot be given no explicit value (always DCDAM_EXCLUSIVE).

    DCDAM_EXCLUSIVE: Lock is enabled.

    DCDAM_NOEXCLUSIVE: Lock is not enabled.

- Flag 3

  Specify whether the function is to wait for the resource to be released from lock if a lock error occurs. This item cannot be specified together with DCDAM_NOEXCLUSIVE. If file-based lock is specified as the type of lock in the function dc_dam_open() in which the file descriptor is specified for damfd, the value specified for this option is meaningless.

  DCDAM_WAIT: The function waits for the resource to be released from lock.

  DCDAM_NOWAIT: The function does not wait for the resource to be released from lock, and returns with an error.

If both items are omitted, the subsequent processing is as follows:

- If DCDAM_WAIT is specified in the function dc_dam_open(), the function waits for the resource to be released from lock.
- If DCDAM_NOWAIT is specified in the function dc_dam_open() or it is omitted, the function returns with an error.

The table below shows the correspondence between flag values specified for flags and the specified type of lock.

Flag 1	Flag 2#1	Flag 3#2	Lock Specified for Flags
REFERENCE	EXCLUSIVE	WAIT	Input for reference, lock used, and waiting for release from lock if a lock error occurs
		NOWAIT	Input for reference, lock used, and error return if a lock error occurs
	NOEXCLUSIVE	N/A	Input for reference, and lock not used#3
MODIFY	N/A	WAIT	Input for update, and waiting for release from lock if a lock error occurs
		NOWAIT	Input for update, and error return if a lock error occurs

Legend:

N/A: Cannot be specified.

#1: The default is NOEXCLUSIVE.

#2: The default is the type of lock specified in the function dc_dam_open().

#3: For a recoverable DAM file, the function dc_dam_read() can be called from a process out of the transaction range only if flag 1 is given the value DCDAM_REFERENCE and flag 2 is given the value DCDAM_NOEXCLUSIVE or is omitted. If the function dc_dam_read() is called with other values specified for the flags from outside the transaction range, it returns with a DCDAMER_PROTO error.

Return values

Return value	Return value (numeric)	Explanation
DC_OK	0	All blocks were input normally.
DCDAMER_PROTO	-1600	The function dc_rpc_open() is not called.
		The purpose of input is updating or lock-specified reference outside the transaction range. (This value is returned only when a recoverable DAM file is accessed.)
		N is specified for atomic_update in the user service definition. (This value is returned only when a recoverable DAM file is accessed.)
		The function dc_dam_start() is not called. (This value is returned only when an unrecoverable DAM file is accessed.)
		The UAP is incorrectly linked as follows: • The library (-ltdam) to be used for access to a TAM file using a DAM service function is linked incorrectly. • The definition of the resource manager for transaction control object files is incorrect.
DCDAMER_EXCER	-1602	A lock error occurred.
DCDAMER_BADF	-1603	The file descriptor specified for damfd is not the one which was acquired by opening the file normally, or the specified file is not open.
DCDAMER_BUFER	-1604	The specified input buffer is too small to contain all blocks.
		The value specified for the input buffer length is outside the range of values that can be specified.
DCDAMER_BNOER	-1606	The relative block number is invalid.
DCDAMER_NOMEM	-1607	The memory became insufficient.
DCDAMER_PARAM_KEYNO	-1609	The value specified for keyno is smaller than 1.
DCDAMER_PARAM_FLAGS	-1611	The value specified for flags is invalid.
DCDAMER_VERSION	-1618	The version of the DAM library linked to the UAP does not allow the UAP to operate with the current DAM service.

Return value	Return value (numeric)	Explanation
DCDAMER_JNLOV	-1613	The number of block updates exceeded the maximum number of blocks that can be updated during one transaction according to the DAM service definition. (Returned only when a recoverable DAM file is accessed.)
DCDAMER_IOER	-1620	An input error occurred.
DCDAMER_LHOLD	-1621	The file of the file descriptor specified for damfd is in logical shutdown state.
DCDAMER_OHOLD	-1622	The file with the file descriptor specified for damfd is in shutdown state due to an error.
DCDAMER_ACCESS	-1628	A DAM file to be accessed is protected with the security facility. The UAP that called the function dc_dam_read() has no access permission.
DCDAMER_TMERR	-1629	An error occurred in the transaction service. (This value is returned only when a recoverable DAM file is accessed.)
DCDAMER_DLOCK	-1642	A deadlock occurred.
DCDAMER_TIMOUT	-1643	The resource could not be acquired because a timeout occurred (the wait time specified in the lock service definition was exceeded).
DCDAMER_LCKOV	-1645	The number of lock requests exceeds the specified maximum number of concurrent lock requests.
DCDAMER_ACSOV	-1648	The maximum number of blocks for access to unrecoverable DAM files is exceeded. (This value is returned only when an unrecoverable DAM file is accessed.)

dc_dam_release - Release a logical file from the shutdown state

Format

- ANSI C, C++

```
#include <dcdam.h>
int  dc_dam_release (char *lfname, DCLONG flags)
```

- K&R C

```
#include <dcdam.h>
int  dc_dam_release (lfname, flags)
char     *lfname;
DCLONG   flags;
```

Description

The function `dc_dam_release()` releases a logical file which has been logically shut down by the function `dc_dam_hold()`. The function `dc_dam_release()` also releases a logical file which has been shut down due to an error.

Arguments whose values are set in the UAP

- `lfname`

  Within 1 to 8 bytes, specify the name of a logical file which is released from the shutdown state.

- `flags`

  Specify the type of release from the shutdown state.

  `DCDAM_LOGICAL_RELEASE`: A file logically shut down is released.

  `DCDAM_OBSTACLE_RELEASE`: A file shut down due to an error is released.

Return values

Return value	Return value (numeric)	Explanation
`DC_OK`	`0`	The logical file specified for `lfname` was released from the shutdown state normally.
`DCDAMER_PROTO`	`-1600`	The function `dc_rpc_open()` is not called.
		`N` is specified for `atomic_update` in the user service definition. (This value is returned only when a recoverable DAM file is accessed.)

Return value	Return value (numeric)	Explanation
		The function `dc_dam_start()` is not called. (This value is returned only when an unrecoverable DAM file is accessed.)
		The UAP is incorrectly linked as follows: • The library (`-ltdam`) to be used for access to a TAM file using a DAM service function is linked incorrectly. • The definition of the resource manager for transaction control object files is incorrect.
DCDAMER_UNDEF	-1601	The logical file specified for `lfname` has not been defined.
DCDAMER_NOMEM	-1607	The memory became insufficient.
DCDAMER_PARAM_LFNAME	-1610	The logical file name specified for `lfname` is invalid.
DCDAMER_PARAM_FLAGS	-1611	The value specified for `flags` is invalid.
DCDAMER_VERSION	-1618	The version of the DAM library linked to the UAP does not allow the UAP to operate with the current DAM service.
DCDAMER_NOEXIST	-1619	The physical file corresponding to the logical file specified for `lfname` does not exist.
DCDAMER_IOER	-1620	An input error occurred.
DCDAMER_NOLHOLD	-1623	The logical file specified for `lfname` is not in logical shutdown state.
DCDAMER_NOOHOLD	-1624	The logical file specified for `lfname` is not in shutdown state due to an error.
DCDAMER_OPENNUM	-1627	The number of open character special files exceeds the specified maximum number.
DCDAMER_ACCESS	-1628	The access permission for character special files has not been granted.
		A DAM file to be accessed is protected with the security facility. The UAP that called the function `dc_dam_release()` has no access permission.
DCDAMER_LFNMER	-1632	The physical file is not a character special file, or the device corresponding to the specified special file does not exist.

Return value	Return value (numeric)	Explanation
DCDAMER_LNOINT	-1633	The physical file corresponding to the logical file specified for lfname has not been initialized as a OpenTP1 file system.
DCDAMER_ACCESSF	-1638	The access permission for the physical file that corresponds to the logical file specified for lfname has not been granted.
DCDAMER_NO_ACL	-1646	A DAM file to be released from shutdown is protected with the security facility. There is no ACL for the corresponding file.

dc_dam_rewrite - Update a logical file block

Format

■ ANSI C, C++

```
#include <dcdam.h>
int  dc_dam_rewrite (int damfd, struct DC_DAMKEY *keyptr,
                     int keyno, char *bufadr, int bufsize,
                     DCLONG flags)
```

■ K&R C

```
#include <dcdam.h>
int  dc_dam_rewrite (damfd, keyptr, keyno, bufadr, bufsize,
                     flags)
int       damfd;
struct DC_DAMKEY *keyptr;
int       keyno;
char      *bufadr;
int       bufsize;
DCLONG    flags;
```

Description

The function `dc_dam_rewrite()` outputs a block, input from the logical file for update processing, a block input by the function `dc_dam_read()`. It also cancels an update request. Block updating timing is shown below.

- For recoverable DAM files

  The updated data is stored in the part of shared memory that is allocated for DAM service, and the actual file is updated when the transaction is committed. A DAM file with deferred update specified is updated asynchronously with the transaction commitment.

- Unrecoverable DAM files

  A DAM file is updated when the function `dc_dam_rewrite()` is returned.

When multiple blocks are specified at a time and if even one of the specified blocks causes an error, processing is stopped and an error is returned. Update processing is not done in this case.

When updating a logical file block, specify the file descriptor which is the return value of the function `dc_dam_open()`.

Arguments whose values are set in the UAP

- `damfd`

  Specify the file name with the file descriptor.

- `keyptr`

  Specify the address of the structure (DAM key) that indicates the block update range. For the structure, specify the block range with the first relative block number and the last relative block number. The structure format is as shown below.

```
struct DC_DAMKEY {
  int fstblkno;
  int endblkno;
  };
```

  `fstblkno`

  Specify the first relative block number of the block to be updated.

  `endblkno`

  Specify the last relative block number of the block to be updated. If 0 is specified, only the block of the relative block number specified for `fstblkno` is updated.

- `keyno`

  Specify the number of structures (number of structure arrays) to be set for `keyptr`.

- `bufadr`

  Specify the address of the update data.

- `bufsize`

  Specify the length of the update data. The length must be (block length to be updated x number of blocks to be updated). You can specify a value in the range from 504 to 2147483647.

- `flags`

  Specify one of the following values as the update request type:

  `DCDAM_UPDATE`

  Update request

  `DCDAM_CANCEL`

  Cancellation of update request

Return values

Return value	Return value (numeric)	Explanation
DC_OK	0	All blocks were updated normally.
DCDAMER_PROTO	-1600	The function dc_rpc_open() is not called.
		The function dc_dam_rewrite() is called outside the transaction range. (This value is returned only when a recoverable DAM file is accessed.)
		N is specified for atomic_update in the user service definition. (This value is returned only when a recoverable DAM file is accessed.)
		The function dc_dam_start() is not called. (This value is returned only when an unrecoverable DAM file is accessed.)
		The UAP is incorrectly linked as follows: • The library (-ltdam) to be used for access to a TAM file using a DAM service function is linked incorrectly. • The definition of the resource manager for transaction control object files is incorrect.
DCDAMER_BADF	-1603	The file descriptor specified for damfd is not the one which was acquired by opening the file normally, or the specified file is not open.
DCDAMER_BUFER	-1604	The update data length (block length to be updated x number of blocks to be updated) is too short.
		The value specified for the update data length is outside the range of values that can be specified.
DCDAMER_SEQER	-1605	The function dc_dam_read() for update processing was not called.
DCDAMER_BNOER	-1606	The relative block number is invalid.
DCDAMER_NOMEM	-1607	The memory became insufficient.
DCDAMER_PARAM_KEYNO	-1609	The value specified for keyno is smaller than 1.
DCDAMER_PARAM_FLAGS	-1611	The value specified for flags is invalid.
DCDAMER_IOER	-1620	An output error occurred. (This value is returned only when an unrecoverable DAM file is accessed.)

Return value	Return value (numeric)	Explanation
DCDAMER_JNLOV	-1613	The number of block updates exceeded the maximum number of blocks that can be updated during one transaction according to the DAM service definition.
DCDAMER_LHOLD	-1621	The file specified for damfd is in logical shutdown state.
DCDAMER_OHOLD	-1622	The file specified for damfd is in shutdown state due to an error.
DCDAMER_TMERR	-1629	An error occurred in the transaction service.
DCDAMER_BUFOV	-1641	The update data length (block length to be updated x number of blocks to be updated) is too long.

dc_dam_start - Start using an unrecoverable DAM file

Format

■ ANSI C, C++

```
#include <dcdam.h>
int  dc_dam_start (DCLONG flags)
```

■ K&R C

```
#include <dcdam.h>
int  dc_dam_start (flags)
DCLONG     flags;
```

Description

The function `dc_dam_start()` declares that an unrecoverable DAM file is used. Call the function `dc_dam_start()` before the function `dc_dam_open()`. Call the function `dc_dam_start()` for each UAP process.

When the function `dc_dam_start()` returns normally, the environment to access an unrecoverable DAM file is established.

Arguments whose value is set in the UAP

■ `flags`

Specify `DCNOFLAGS`.

Return values

Return value	Return value (numeric)	Explanation
`DC_OK`	`0`	Normal termination. Unrecoverable DAM files now can be used.
`DCDAMER_PROTO`	`-1600`	The function `dc_rpc_open()` is not called.
`DCDAMER_NOMEM`	`-1607`	The memory became insufficient.
`DCDAMER_PARAM_FLAGS`	`-1611`	The value specified for `flags` is invalid.
`DCDAMER_VERSION`	`-1618`	The UAP is linked with a DAM library which is inoperable with the current DAM service.
`DCDAMER_STARTED`	`-1647`	The function `dc_dam_start()` has already been called.

dc_dam_status - Reference the status of a logical file

Format

- ANSI C, C++

```
#include <dcdam.h>
int  dc_dam_status (char *lfname, struct DC_DAMSTAT *stbuf,
                    int reserve, DCLONG flags)
```

- K&R C

```
#include <dcdam.h>
int  dc_dam_status (lfname, stbuf, reserve, flags)
char      *fname;
struct  DC_DAMSTAT  *stbuf;
int       reserve;
DCLONG    flags;
```

Description

The function `dc_dam_status()` returns the current logical file status to structure `DC_DAMSTAT`. The following values are returned:

- Number of blocks in a logical file
- Logical file block length
- Physical file name corresponding to the logical file
- Current logical file status (whether it is shut down)
- Attribute of the logical file specified in the DAM service definition
- Security attribute of the logical file specified in the DAM service definition

The function `dc_dam_status()` can be called before and after a logical file is opened with the function `dc_dam_open()`.

When referencing the status of a logical file, specify the logical file name.

Arguments whose values are set in the UAP

- `lfname`

  Specify a logical file name within eight bytes.

- `stbuf`

  Specify the address of a structure `DC_DAMSTAT` that receives the logical file status. The logical file status set in the function `dc_dam_status()` is returned in the structure.

- phyfilno

  Area used by the DAM service. Specify null character (0).

- flags

  Specify DCNOFLAGS.

Argument whose value is returned from OpenTP1

- stbuf

  The logical file status data is returned in the format of structure DC_DAMSTAT as follows:

```
struct  DC_DAMSTAT {
                    int   st_block_len;
                    int   st_block_num;
                    char  st_file_ph_name[64];
                    char  st_file_stat;
                    char  st_file_def;
                    char  st_file_sec;
                    char  st_filler_1;
                    char  st_file_inf;
                    };
```

  - st_block_len

    The block length of a logical file is returned.

  - st_block_num

    The number of blocks in a logical file is returned.

  - st_file_ph_name

    The physical file name corresponding to the logical file is returned.

  - st_file_stat

    The current logical file status is returned as follows:

    DCDAM_ST_NOT_HOLD: The logical file can be accessed.

    DCDAM_ST_HOLD_LOG: The logical file is logically shut down.

    DCDAM_ST_HOLD_OBS: The logical file is shut down with an error.

    DCDAM_ST_HOLD_REQ: A shutdown request is being made for the logical file.

  - st_file_def

    The attribute of the logical file specified in the DAM service definition is returned as follows:

    DCDAM_ST_QUICK: DAM file ineligible for deferred update processing

DCDAM_ST_DEFERRED: DAM file eligible for deferred update processing

DCDAM_ST_NORECOVER: Unrecoverable DAM file

DCDAM_ST_CACHELESS: Unrecoverable DAM file specified by a cache-less access

- st_file_sec

  The security attribute of the logical file specified in the DAM service definition is returned as follows:

  DCDAM_ST_NON: Security is not specified.

  DCDAM_ST_SEC: Security is specified.

- st_filler_1

  Reserved area 1 (A null character (0) is set.)

- st_file_inf

  Reserved area 2 (The value -1 is set.)

Return values

Return value	Return value (numeric)	Explanation
DC_OK	0	The logical file status is set normally in the structure DC_DAMSTAT.
DCDAMER_PROTO	-1600	The function dc_rpc_open() is not called.
		N is specified for atomic_update in the user service definition. (This value is returned only when a recoverable DAM file is accessed.)
		The function dc_dam_start() is not called. (This value is returned only when an unrecoverable DAM file is accessed.)
DCDAMER_UNDEF	-1601	The logical file name specified for lfname is undefined.
DCDAMER_NOMEM	-1607	The memory became insufficient.
DCDAMER_PARAM_LFNAME	-1610	The logical file name specified for lfname is invalid.
DCDAMER_PARAM_FLAGS	-1611	The value specified for flags is invalid.
DCDAMER_PARAM_ERROR	-1612	The value specified for the argument is invalid.
		The value specified for stbuf is invalid.

Return value	Return value (numeric)	Explanation
		No null character is set for reserve.
DCDAMER_VERSION	-1618	The UAP is linked with a DAM library which is inoperable with the current DAM service.
DCDAMER_ACCESS	-1628	A DAM file whose status is to be referenced is protected with the security facility. The UAP that called the function dc_dam_status() has no access permission.
DCDAMER_NO_ACL	-1646	A DAM file whose status is to be referenced is protected with the security facility. There is no ACL for the corresponding file.

Note

When the function dc_dam_status() is called, the DAM service does lock control to get data. So if the function dc_dam_status() is used too often, the throughput may be degraded because of the lock release wait time. Therefore, reference the DAM file status as little as possible while online.

dc_dam_write - Output a logical file block

Format

- ANSI C, C++

```
#include <dcdam.h>
int  dc_dam_write (int damfd, struct DC_DAMKEY *keyptr,
                   int keyno, char *bufadr, int bufsize,
                   DCLONG flags)
```

- K&R C

```
#include <dcdam.h>
int  dc_dam_write (damfd, keyptr, keyno, bufadr, bufsize,
                   flags)
int       damfd;
struct DC_DAMKEY *keyptr;
int       keyno;
char      *bufadr;
int       bufsize;
DCLONG    flags;
```

Description

The function `dc_dam_write()` outputs a specified block. The block output timing is given below.

- For recoverable DAM files

  The updated data is stored in the part of shared memory that is allocated for DAM service, and the actual file is updated when the transaction is committed. A DAM file with deferred output specified is output asynchronously with the transaction commitment.

- Unrecoverable DAM files

  A DAM file is output when the function `dc_dam_write()` returns.

When a request is made to output multiple blocks at a time and if even one of the specified blocks causes an error, processing is stopped and an error is returned. The blocks are not output in this case.

When outputting a logical file block, specify the file descriptor which is the return value of the function `dc_dam_open()`.

Arguments whose values are set in the UAP

- `damfd`

  Specify the file descriptor of the file to which a block is output.

- keyptr

  Specify the address of the structure (DAM key) that indicates the block output range. For the structure, specify the block range with the first relative block number and the last relative block number. The structure format is as shown below.

```
struct DC_DAMKEY {
                  int fstblkno;
                  int endblkno;
                 };
```

  - fstblkno

    Specify the first relative block number of the block to be output.

  - endblkno

    Specify the last relative block number of the block to be output. If 0 is specified, only the block of the relative block number specified for fstblkno is updated.

- keyno

  Specify the number of structures (number of structure arrays) to be set for keyptr.

- bufadr

  Specify the address of the update data.

- bufsize

  Specify the length of the output data. The length must be (block length to be output x number of blocks to be output). You can specify a value in the range from 504 to 2147483647.

- flags

  Specify whether the function is to wait for the resource to be released from lock if a lock error occurs.

  DCDAM_WAIT: The function waits for the resource to be released from lock.

  DCDAM_NOWAIT: The function does not wait for the resource to be released from lock, and returns with an error.

  DCNOFLAGS: Processing is done according to the value specified for flags of the function dc_dam_open().

  If DCNOFLAGS is specified, the subsequent processing is as follows:

  - If DCDAM_WAIT is specified in the function dc_dam_open(), the function waits for the resource to be released from lock.
  - If DCDAM_NOWAIT is specified in the function dc_dam_open() or it is omitted, the function returns with an error.

If the function dc_dam_open() in which the file descriptor is specified for damfd specifies files-based lock as the type of lock, the value specified for this option is meaningless.

Return values

Return value	Return value (numeric)	Explanation
DC_OK	0	All blocks were output normally.
DCDAMER_PROTO	-1600	The function dc_rpc_open() is not called.
		The function dc_dam_write() is called outside the transaction range. (This value is returned only when a recoverable DAM file is accessed.)
		N is specified for atomic_update in the user service definition. (This value is returned only when a recoverable DAM file is accessed.)
		The function dc_dam_start() is not called. (This value is returned only when an unrecoverable DAM file is accessed.)
		The UAP is incorrectly linked as follows: • The library (-ltdam) to be used for access to a TAM file using a DAM service function is linked incorrectly. • The definition of the resource manager for transaction control object files is incorrect.
DCDAMER_EXCER	-1602	A lock specification error occurred.
DCDAMER_BADF	-1603	The file descriptor specified for damfd is not the one which was acquired by opening the file normally, or the specified file has not been defined.
DCDAMER_BUFER	-1604	The output data length (block length to be output x number of blocks to be output) is too short.
		The value specified for the output data length is outside the range of values that can be specified.
DCDAMER_SEQER	-1605	The call sequence of functions is invalid.
DCDAMER_BNOER	-1606	The relative block number is invalid.
DCDAMER_NOMEM	-1607	The memory became insufficient.
DCDAMER_PARAM_KEYNO	-1609	The value specified for keyno is smaller than 1.
DCDAMER_PARAM_FLAGS	-1611	The value specified for flags is invalid.

Return value	Return value (numeric)	Explanation
DCDAMER_JNLOV	-1613	The number of block updates exceeded the maximum number of blocks that can be updated during one transaction according to the DAM service definition. (Returned only when a recoverable DAM file is accessed).
DCDAMER_IOER	-1620	An output error occurred. (This value is returned only when an unrecoverable DAM file is accessed.)
DCDAMER_LHOLD	-1621	The file specified for damfd is in logical shutdown state.
DCDAMER_OHOLD	-1622	The file specified for damfd is in shutdown state due to an error.
DCDAMER_ACCESS	-1628	A DAM file to be accessed is protected with the security facility. The UAP that called the function dc_dam_write() has no access permission.
DCDAMER_TMERR	-1629	An error occurred in the transaction service. (This value is returned only when a recoverable DAM file is accessed.)
DCDAMER_BUFOV	-1641	The output data length (block length to be output x number of blocks to be output) is too long.
DCDAMER_DLOCK	-1642	A deadlock occurred.
DCDAMER_TIMOUT	-1643	The resource could not be acquired because a timeout occurred (the wait time specified in the lock service definition was exceeded).
DCDAMER_LCKOV	-1645	The number of lock requests exceeds the specified maximum number of concurrent lock requests.
DCDAMER_ACSOV	-1648	The maximum number of blocks that can be accessed is exceeded. (This value is returned only when an unrecoverable DAM file is accessed.)

Note

Do the following if the values DCDAMER_JNLOV and DCDAMER_ACSOV are returned:

- Set the number of output blocks to the same or less than the maximum number of blocks that can be updated.
- If there is a block that has not been updated with the function dc_dam_rewrite(), update it before calling the function dc_dam_write().

IST service (dc_ist_~)

This section explains functions that access an internode shared table. The syntax of the following functions are explained:

- `dc_ist_close` - Close an internode shared table
- `dc_ist_open` - Open an internode shared table
- `dc_ist_read` - Input an internode shared table record
- `dc_ist_write` - Output an internode shared table record

The functions for IST service (`dc_ist_~`) can be used only in UAPs of TP1/Server Base. They cannot be used in UAPs of TP1/LiNK.

dc_ist_close - Close an internode shared table

Format

■ ANSI C, C++

```
#include <dcist.h>
int  dc_ist_close (int istid, DCLONG flags)
```

■ K&R C

```
#include <dcist.h>
int  dc_ist_close (istid, flags)
int     istid;
DCLONG  flags;
```

Description

The function `dc_ist_close()` closes a specified internode shared table.

Arguments whose values are set in the UAP

■ `istid`

Specify the table descriptor of the internode shared table to be closed.

■ `flags`

Specify `DCNOFLAGS`.

Return values

Return value	Return value (numeric)	Explanation
`DC_OK`	`0`	The internode shared table was closed normally.
`DCISTER_PROTO`	`-3800`	The call sequence of functions which access the internode shared table is invalid.
`DCISTER_BADID`	`-3803`	The table descriptor specified for `istid` is not the one which was acquired by opening the table normally.
		The internode shared table is not open.
`DCISTER_PARAM_FLAGS`	`-3811`	The value specified for `flags` is invalid.

dc_ist_open - Open an internode shared table

Format

- ANSI C, C++

```
#include <dcist.h>
int  dc_ist_open (char *istname, DCLONG flags)
```

- K&R C

```
#include <dcist.h>
int  dc_ist_open (istname, flags)
char    *istname;
DCLONG  flags;
```

Description

The function `dc_ist_open()` opens a specified internode shared table. When an internode shared table is opened normally, a table descriptor is returned.

Arguments whose values are set in the UAP

- `istname`

  Specify the internode shared table name to be opened within eight bytes.

- `flags`

  Specify `DCNOFLAGS`.

Return values

Return value	Return value (numeric)	Explanation
0 or positive integer		0 or positive integer indicates a table descriptor.
`DCISTER_PROTO`	`-3800`	The call sequence of functions which access the internode shared table is invalid.
`DCISTER_UNDEF`	`-3801`	The internode shared table name specified for `istname` is undefined.
`DCISTER_NOMEM`	`-3807`	The memory became insufficient.
`DCISTER_OPENED`	`-3808`	The name of an already open internode shared table was specified for `istname`.
`DCISTER_PARAM_TBLNAME`	`-3810`	The length of the value specified for the internode shared table name is invalid.

Return value	Return value (numeric)	Explanation
DCISTER_PARAM_FLAGS	-3811	The value specified for flags is invalid.

dc_ist_read - Input an internode shared table record

Format

- ANSI C, C++

```
#include <dcist.h>
int  dc_ist_read (int istid, struct DC_ISTKEY *keyptr,
                  int keyno, char *bufadr, int bufsize,
                  DCLONG flags)
```

- K&R C

```
#include <dcist.h>
int  dc_ist_read (istid, keyptr, keyno, bufadr, bufsize,
                  flags)
int        istid;
struct  DC_ISTKEY  *keyptr;
int        keyno;
char       *bufadr;
int        bufsize;
DCLONG     flags;
```

Description

The function `dc_ist_read()` inputs a record in a specified range from a specified internode shared table. If multiple records are collectively specified and an error occurs with any of the specified records, the function `dc_ist_read()` returns with an error without inputting the records to the input buffer.

When inputting an internode shared table record, specify the table descriptor which is the return value of the function `dc_ist_open()`.

Arguments whose values are set in the UAP

- `istid`

  Specify the table descriptor of the internode shared table to be accessed.

- `keyptr`

  Specify the address of the structure (IST key) indicating the range of the relative record numbers of the record to be referenced. For the structure, specify the record range with the first and last relative block numbers. The structure formats are as follows:

```
struct  DC_ISTKEY {
                   int  fstrecno;
                   int  endrecno;
                   };
```

  - `fstrecno`

Specify the first relative record number of the record to be accessed.

- `endrecno`

  Specify the last relative record number of the record to be accessed. If 0 is specified, only the record with the relative record number specified with `fstrecno` is input.

- `keyno`

  Specify the number of structures (number of arrays in the structure) to be specified for `keyptr`.

- `bufadr`

  Specify the input buffer address.

- `bufsize`

  Specify the input buffer length. The value must be (input record length x number of input records) or greater.

- `flags`

  Specify `DCNOFLAGS`.

Return values

Return value	Return value (numeric)	Explanation
`DC_OK`	`0`	All specified records are normally input.
`DCISTER_PROTO`	`-3800`	The call sequence of functions which access the internode shared table is invalid.
`DCISTER_BADID`	`-3803`	The table descriptor specified for `istid` is not the one which was acquired by opening the table normally.
		The internode shared table is not open.
`DCISTER_BUFER`	`-3804`	The input buffer length specified for `bufsize` is insufficient to input all records.
`DCISTER_RNOER`	`-3806`	The relative record number is invalid.
`DCISTER_NOMEM`	`-3807`	The memory became insufficient.
`DCISTER_PARAM_KEYNO`	`-3809`	The value specified for `keyno` is less than 1.
`DCISTER_PARAM_FLAGS`	`-3811`	The value specified for `flags` is invalid.

dc_ist_write - Output an internode shared table record

Format

- ANSI C, C++

```
#include <dcist.h>
int  dc_ist_write (int istid, struct DC_ISTKEY *keyptr,
                   int keyno, char *bufadr, int bufsize,
                   DCLONG flags)
```

- K&R C

```
#include <dcist.h>
int  dc_ist_write (istid, keyptr, keyno, bufadr, bufsize,
                   flags)
int        istid;
struct  DC_ISTKEY  *keyptr;
int        keyno;
char       *bufadr;
int        bufsize;
DCLONG     flags;
```

Description

The function `dc_ist_write()` outputs a record in a specified range to an internode shared table. If multiple records are collectively specified and an error occurs with any of the specified records, the function `dc_ist_write()` returns with an error without outputting the records to the output buffer.

When the function `dc_ist_write()` terminates normally, the contents of the record at the local node are updated. The contents of internode shared tables at other nodes are updated with a certain time interval after this function returns normally.

When outputting an internode shared table record, specify the table descriptor which is the return value of the function `dc_ist_open()`.

Arguments whose values are set in the UAP

- `istid`

  Specify the table descriptor of the internode shared table to be accessed.

- `keyptr`

  Specify the address of the structure (IST key) indicating the range of the relative record numbers of the record to be output. For the structure, specify the record range with the first and last relative block numbers. The structure formats are as follows:

```
struct  DC_ISTKEY {
                  int  fstrecno;
                  int  endrecno;
                  };
```

- `fstrecno`

  Specify the first relative record number of the record to be accessed.

- `endrecno`

  Specify the last relative record number of the record to be accessed. If 0 is specified, only the record with the relative record number specified with `fstrecno` is input.

- `keyno`

  Specify the number of structures (number of arrays in the structure) to be specified for `keyptr`.

- `bufadr`

  Specify the address of the buffer containing update data to be output.

- `bufsize`

  Specify the output buffer length. The value must be (output record length x number of output records).

- `flags`

  Specify `DCNOFLAGS`.

Return values

Return value	Return value (numeric)	Explanation
`DC_OK`	`0`	All specified records are normally output.
`DCISTER_PROTO`	`-3800`	The call sequence of functions which access the internode shared table is invalid.
`DCISTER_BADID`	`-3803`	The table descriptor specified for `istid` is not the one which was acquired by opening the table normally.
		The internode shared table is not open.
`DCISTER_BUFER`	`-3804`	The output buffer length specified for `bufsize` is insufficient to output all records.
`DCISTER_RNOER`	`-3806`	The relative record number is invalid.
`DCISTER_NOMEM`	`-3807`	The memory became insufficient.

Return value	Return value (numeric)	Explanation
DCISTER_PARAM_KEYNO	-3809	The value specified for keyno is less than 1.
DCISTER_PARAM_FLAGS	-3811	The value specified for flags is invalid.
DCISTER_BUFOV	-3841	The output buffer length is greater than the total length of records to be output.

User journal acquisition (dc_jnl_~)

This section gives the syntax and other information of the following function which is used for acquiring user journals:

- `dc_jnl_ujput` - Acquire a user journal

The function for user journal acquisition (`dc_jnl_~`) can be used only in UAPs of TP1/Server Base. They cannot be used in UAPs of TP1/LiNK.

dc_jnl_ujput - Acquire a user journal

Format

- ANSI C, C++

```
#include <dcjnl.h>
int  dc_jnl_ujput (char *data, DCULONG dsize,
                   DCLONG ujcode, DCLONG flags)
```

- K&R C

```
#include <dcjnl.h>
int  dc_jnl_ujput (data, dsize, ujcode, flags)
char      *data;
DCULONG   dsize;
DCLONG    ujcode;
DCLONG    flags;
```

Description

The function `dc_jnl_ujput()` acquires a user journal (UJ), which is UAP historical information, into the system journal file (`system_jnl_file`). The unit of UJ acquired by calling the function `dc_jnl_ujput()` once is called an UJ record.

A user journal is not output to the system journal file immediately after the function `dc_jnl_ujput()` is called. The UJ record is output to the system journal file when the journal buffer becomes full or when the synchronization point at which the transaction processing terminated normally is acquired.

The function `dc_jnl_ujput()` can be called after the function `dc_rpc_open()` has been called and before the function `dc_rpc_close()` is called. Even if an error occurs in the transaction processing that called the function `dc_jnl_ujput()`, the UJ record that has already been output cannot be invalidated through rollback processing (partial recovery). Even when rollback processing is executed for the transaction processing that called the function `dc_jnl_ujput()`, the UJ record is output to the system journal file.

Arguments whose values are set in the UAP

- `data`

  Specify the UAP historical information to be acquired. Data valid as UAP historical information must be as long as specified for dsize.

- `dsize`

  Specify the length of the UAP historical information to be acquired. The specified length must be in the range from 1 to (the value specified for the `jnl_max_datasize`

operand of the system journal file service definition at the acquisition destination - 8).

- `ujcode`

  Specify the UJ code as a value from 0 to 255.

- `flags`

  Using one of the following values, specify whether to output the UJ record to the system journal file at acquisition of the UJ record.

  `DCJNL_FLUSH`

  Output the UJ record to the system journal file at acquisition of the UJ record. If the UJ record is acquired inside the transaction, this setting is ignored.

  `DCNOFLAGS`

  Do not output the UJ record to the system journal file at acquisition of the UJ record.

Return values

Return value	Return value (numeric)	Explanation
`DC_OK`	`0`	Normal termination.
`DCJNLER_PARAM`	`-1101`	The parameter format is invalid.
`DCJNLER_SHORT`	`-1102`	The value specified for the length of user journal (`dsize` value) is 0 or less.
`DCJNLER_LONG`	`-1103`	The value specified for the length of user journal (`dsize` value) exceeds the limit.
`DCJNLER_PROTO`	`-1105`	The `dc_rpc_open()` function has not been called. Or, the `dc_jnl_ujput()` function cannot be used because the execution environment of the applicable system is in journal fileless mode.

Note

A UJ record that is outside the transaction is output to the system journal file when the journal buffer becomes full or when a transaction of another application terminates normally (when the transaction processing is committed). To acquire the UJ record using an application that does not generate transactions, call the function `dc_jnl_ujput()` in which `DCJNL_FLUSH` is set for `flags` at the appropriate timing.

Lock for resources (dc_lck_~)

This section gives the syntax and other information of the following functions which are used for locking arbitrary user files:

- `dc_lck_get` - Enable locking of a resource
- `dc_lck_release_all` - Release all the resources from lock
- `dc_lck_release_byname` - Release resource from lock specified by name

The functions for lock for resources (`dc_lck_~`) can be used only in UAPs of TP1/Server Base. They cannot be used in UAPs of TP1/LiNK.

dc_lck_get - Enable locking of a resource

Format

- ANSI C, C++

```
#include <dclck.h>
int  dc_lck_get (char *name, DCLONG lockmode,
                 DCLONG ownerflag, DCLONG flags)
```

- K&R C

```
#include <dclck.h>
int  dc_lck_get (name, lockmode, ownerflag, flags)
char      *name;
DCLONG    lockmode;
DCLONG    ownerflag;
DCLONG    flags;
```

Description

The function `dc_lck_get()` specifies lock for resources to be used by UAPs. Lock is managed in global transactions which are managed by the OpenTP1 transaction manager.

The lock specified by the function `dc_lck_get()` is released by lock release function (`dc_lck_release_all()` or `dc_lck_release_byname()`). The lock is also released when the synchronization point of the global transaction that called the function `dc_lck_get()` is acquired.

Arguments whose values are set in the UAP

- `name`

  Specify the name of the resource for which lock is to be specified. The name can be specified with up to 16-byte alphanumeric characters. The OpenTP1 lock service manages the lock on the basis of the specified resource name. If a value less than 16 bytes is specified and a null character appears, the value before the null character is regarded as the resource name. If a value exceeding 16 bytes is specified, the value up to 16 bytes is regarded as the resource name. The excess bytes are truncated.

  The lock service does not check the contents of the character string. Specify a logically correct name. If a value other than alphanumeric characters is used for a resource name, the deadlock information, the timeout information, and the `lckls` command might not be displayed normally.

- `lockmode`

  Specify a lock mode. The lock mode must be `DCLCK_PR` or `DCLCK_EX`. They cannot

be specified at the same time.

DCLCK_PR

The resource is referenced. Other UAPs are permitted to reference the resource but are not permitted to update it.

DCLCK_EX

The resource is updated. Other UAPs are not permitted to reference or update the resource.

ownerflag

Specify DCLCK_OWNER_MIGRATE.

- flags

Specify a flag concerning lock for the resource. The following values can be specified:

DCLCK_WAIT

If a UAP competes for the resource with another UAP, the UAP waits until the resource is released. If this flag is not set when UAPs compete for the resource, an error is returned.

DCLCK_TEST

Specify this flag to check whether the resource can be used. Note the following even if the function dc_lck_get() terminates normally when this flag is set:

Lock is not enabled for the resource specified for name.

DCNOFLAGS

No flag is set.

Return values

Return value	Return value (numeric)	Explanation
DC_OK	0	Normal termination.
DCLCKER_PARAM	-401	The value specified for the argument is invalid.
DCLCKER_WAIT	-450	Another UAP is using the resource specified for name.
DCLCKER_DLOCK	-452	A deadlock occurred.
DCLCKER_TIMOUT	-453	The resource could not be acquired because a timeout occurred (the wait time specified in the OpenTP1 lock service definition was exceeded).
DCLCKER_MEMORY	-454	The table for lock is insufficient.

Return value	Return value (numeric)	Explanation
DCLCKER_OUTOFTRN	-455	The specification was made by a UAP which was not operating as a transaction.
DCLCKER_VERSION	-457	The OpenTP1 library version does not match the lock service version.

dc_lck_release_all - Release all the resources from lock

Format

- ANSI C, C++

```
#include <dclck.h>
int  dc_lck_release_all (DCLONG ownerflag, DCLONG flags)
```

- K&R C

```
#include <dclck.h>
int  dc_lck_release_all (ownerflag, flags)
DCLONG     ownerflag;
DCLONG     flags;
```

Description

The function `dc_lck_release_all()` releases all the resources from lock which was specified in the function `dc_lck_get()`. Call the function `dc_lck_release_all()` when releasing the resources from lock before the synchronization point is acquired.

When the global transaction with lock specified terminates, the OpenTP1 lock service automatically releases the resources from lock. In this case, there is no need to specify release from lock in the UAP.

Arguments whose values are set in the UAP

- `ownerflag`

  Specify `DCLCK_OWNER_MIGRATE`.

- `flags`

  Specify `DCNOFLAGS`.

Return values

Return value	Return value (numeric)	Explanation
DC_OK	0	Normal termination.
DCLCKER_PARAM	-401	The value specified for the argument is invalid.
DCLCKER_NOTHING	-456	The resource was not acquired for the transaction that called this function.
DCLCKER_OUTOFTRN	-455	The function was called from a UAP which was not operating as a transaction.

Return value	Return value (numeric)	Explanation
DCLCKER_VERSION	-457	The OpenTP1 library version does not match the lock service version.

dc_lck_release_byname - Release resource from lock specified by name

Format

- ANSI C, C++

```
#include <dclck.h>
int  dc_lck_release_byname (char *name, DCLONG ownerflag,
                            DCLONG flags)
```

- K&R C

```
#include <dclck.h>
int  dc_lck_release_byname (name, ownerflag, flags)
char      *name;
DCLONG    ownerflag;
DCLONG    flags;
```

Description

The function `dc_lck_release_byname()` specifies the name of a resource for which the function `dc_lck_get()` specified lock, and releases the resource from the lock. Call the function `dc_lck_release_byname()` when releasing the resource from lock before the synchronization point is acquired.

When the global transaction with lock specified terminates, the OpenTP1 lock service automatically releases the resource from lock. In this case, there is no need to specify release from lock in the UAP.

Arguments whose values are set in the UAP

- `name`

  Specify the name of the resource to be released from lock. The resource name must be identical to the name specified in the function `dc_lck_get()`.

- `ownerflag`

  Specify `DCLCK_OWNER_MIGRATE`.

- `flags`

  Specify `DCNOFLAGS`.

Return values

Return value	Return value (numeric)	Explanation
DC_OK	0	Normal termination.
DCLCKER_PARAM	-401	The value specified for the argument is invalid.
DCLCKER_NOTHING	-456	The resource that corresponds to the resource name specified for release from lock does not exist.
DCLCKER_OUTOFTRN	-455	The function was called from a UAP which was not operating as a transaction.
DCLCKER_VERSION	-457	The OpenTP1 library version does not match the lock service version.

Audit log output (dc_log_audit_~)

This section gives the syntax and other information of the following functions which are used to output audit log data from a UAP:

- `dc_log_audit_print` - Output audit log data

dc_log_audit_print - output audit log data

Format

- ANSI C, C++

```
#include <dclog.h>
int  dc_log_audit_print(char *msgid,char *compid,DCLONG ctgry,
                        DCLONG result,DCLONG op,char *msg,DCLONG flags)
```

- K&R C

```
#include <dclog.h>
int  dc_log_audit_print(msgid,compid,ctgry,result,op,msg,flags)
char      *msgid;
char      *compid;
DCLONG    ctgry;
DCLONG    result;
DCLONG    op;
char      *msg;
DCLONG    flags;
```

Description

The function `dc_log_audit_print()` outputs to the audit log file the following information items, in addition to the information specified as arguments: header information, serial number, date and time, relevant program name, relevant process ID, location, subject identification information, object information, object location information, request sender host, and location identification information. The *relevant program* means the program that generated the audit log data, which is OpenTP1. If an error occurs during output of audit log data, an error message is sent to the standard error output and syslog.

In OpenTP1, numbers from 34000 to 34999 are assigned for message IDs used by the function `dc_log_audit_print()`. If you create a UAP, make sure that the message IDs output by the UAP are in the range from 34000 to 34999.

For details on the items output as audit log data, see the *OpenTP1 Programming Guide*.

Arguments whose values are set in the UAP

- `msgid`

  Specify an identifier uniquely assigned to each audit log entry (message ID) in the format `KFCA`*nnnnn*`-`*x* (11 characters) and follow the identifier with a null character. For *nnnnn*, specify a five-digit serial number in the range from 34000 to 34999. For *x*, specify `E`, `W`, or `I` as the message type according to the type of information provided by the audit log entry to be output.

- `compid`

  Specify any value that identifies the UAP that called the function `dc_log_audit_print()` (calling program ID). The value you set must be two numeric characters, alphabetic characters, or symbols followed by a null character. In the audit log, the format is `*`*AA*, with an asterisk (`*`) prefixed (*AA*: character string specified in `compid`).

- `ctgry`

  Specify one of the following values as the audit event type:

  `DCLOG_CTG_STARTSTOP`: Audit event related to a start or stop operation

  `DCLOG_CTG_AUTH`: Audit event related to identification or authentication

  `DCLOG_CTG_ACCESS`: Audit event related to access control

  `DCLOG_CTG_CONFIG`: Audit event related to the configuration definition

  `DCLOG_CTG_FAIL`: Audit event related to failures

  `DCLOG_CTG_LINK`: Audit event related to the linkage status

  `DCLOG_CTG_EXTERNAL`: Audit event related to external services

  `DCLOG_CTG_CONTENT`: Audit event related to access to important information

  `DCLOG_CTG_MAINTAIN`: Audit event related to maintenance

  `DCLOG_CTG_ANORMALY`: Audit event related to anomalies

  `DCLOG_CTG_MANAGE`: Audit event related to management operation

  For details on audit event types, see the manual *OpenTP1 Operation*.

- `result`

  Set one of the following values as the audit event result to be included in the audit log data:

  `DCLOG_RES_SUCCESS`: Successful event

  `DCLOG_RES_FAIL`: Failed event

  `DCLOG_RES_OCCUR`: Event that cannot be categorized as success or failure

- `op`

  Specify the value to be included as operation information in the audit log data. Make sure that you specify one of the following reserved words according to the audit event type specified by `ctgry`. If you specify NULL, this item will not be included in the audit log data.

Table 2-1: Correspondence between audit event types and reserved words

Audit event type	Reserved word	Meaning
DCLOG_CTG_STARTSTOP (start or stop operation)	DCLOG_OP_START	Start or activation
	DCLOG_OP_STOP	Termination or stop
DCLOG_CTG_AUTH (identification or authentication)	DCLOG_OP_LOGIN	Login
	DCLOG_OP_LOGOUT	Logout
	DCLOG_OP_LOGON	Logon
	DCLOG_OP_LOGOFF	Logoff
	DCLOG_OP_DISABLE	Account disabled
DCLOG_CTG_ACCESS (access control)	DCLOG_OP_ENFORCE	Enforcement
DCLOG_CTG_CONFIG (configuration definition)	DCLOG_OP_REFER	Reference
	DCLOG_OP_ADD	Addition
	DCLOG_OP_UPDATE	Updating
	DCLOG_OP_DELETE	Deletion
DCLOG_CTG_FAIL (failures)	DCLOG_OP_OCCUR	Occurrence
DCLOG_CTG_LINK (linkage status)	DCLOG_OP_UP	Linkage active
	DCLOG_OP_DOWN	Linkage inactive
DCLOG_CTG_EXTERNAL (external services)	DCLOG_OP_REQ	Request
	DCLOG_OP_RES	Response
	DCLOG_OP_SEND	Sending
	DCLOG_OP_RECV	Receiving
DCLOG_CTG_CONTENT (access to important information)	DCLOG_OP_REFER	Reference
	DCLOG_OP_ADD	Addition
	DCLOG_OP_UPDATE	Updating
	DCLOG_OP_DELETE	Deletion

Audit event type	Reserved word	Meaning
DCLOG_CTG_MAINTAIN (maintenance)	DCLOG_OP_INSTALL	Installation
	DCLOG_OP_UNINSTALL	Uninstallation
	DCLOG_OP_UPDATE	Updating
	DCLOG_OP_BACKUP	Backup
	DCLOG_OP_MAINTAIN	Maintenance work
DCLOG_CTG_ANORMALY (anomalies)	DCLOG_OP_OCCUR	Occurrence
DCLOG_CTG_MANAGE (management operation)	DCLOG_OP_INVOKE	Invocation (the administrator)
	DCLOG_OP_NOTIFY	Notification (the administrator)

- msg

  Specify the address of the area that contains the freely specified description to be included in the audit log data. If you specify NULL, this item will not be included in the audit log data.

  You can use numeric characters, alphabetic characters, symbols, spaces, double quotation marks ("), and commas (,). The description can have a maximum of 1024 characters, and must be followed by a null character. The null terminator character is not included in the number of characters in the description.

  In the log, the specified description is enclosed in double quotation marks ("). If a double quotation mark (") is included in the description, the double quotation mark is prefixed by another double quotation mark.

- flags

  Specify DCNOFLAGS.

Return value

Return value	Return value (numeric)	Explanation
DCLOG_AUDIT_OFF	1	Output of audit log data has been disabled. Possible causes are as follows: • The log_audit_out operand in the log service definition has been set to N or has not been specified. • The log_audit_suppress operand has been set to Y in the log service definition.
		The message ID specified in the msgid argument has not been specified in the log_audit_message operand in the log service definition.
		An invalid message has been specified.
DC_OK	0	The function terminated normally.
DCLOGER_PARAM_ARGS	-1900	The value specified as an argument is incorrect.
DCLOGER_DEFFILE	-1904	Definition analysis failed.
DCLOGER_PROTO	-1999	The dc_rpc_open function was not issued.
DCLOGER_FATAL	-1997	An error other than the above occurred.

Output message log (dc_log~)

This section gives the syntax and other information of the following function which is used for outputting message log from the UAP:

- `dc_logprint` - Output message log

The function for output message log (`dc_log_~`) can be used in UAPs of both TP1/Server Base and TP1/LiNK.

dc_logprint - Output message log

Format

- ANSI C, C++

```
#include <dclog.h>
int  dc_logprint (char *msgid, char *pgm_id, char *string,
                  char *info, DCLONG color, DCLONG flags)
```

- K&R C

```
#include <dclog.h>
int  dc_logprint (msgid, pgm_id, string, info, color,
                  flags)
char      *msgid;
char      *pgm_id;
char      *string;
char      *info;
DCLONG    color;
DCLONG    flags;
```

Description

The function `dc_logprint()` outputs a character string specified for an argument to the message log file. Before the output, the function `dc_logprint()` adds the following information to the character string through OpenTP1:

- Line header
- OpenTP1 ID
- Date and time
- Request source node name
- Request source program ID
- Message ID

OpenTP1 assigns a number from 05000 to 06999 to a message ID used in the function `dc_logprint()`. Assign a number from 05000 to 06999 to a message ID output from a UAP.

Even if an error occurs, `DC_OK` might be returned. Consequently, a message log might be missing. The missing message log can be identified by checking the message log serial numbers.

If the function `dc_logprint()` is called more than once from one process, the sequence of output to the message log file is ensured. However, if the function `dc_logprint()` is called from each of multiple processes, the message logs might

not be output to the message log file in the issue sequence.

If a communication error (DCLOGER_COMM) or a log service inactive error (DCLOGER_NOT_UP) occurs, the message issued from the UAP is edited in the UAP process and is output to the standard error output file. Either of the following codes which indicate the causes of errors is added to the end of the message:

- E1

  Indicates that the message log could not be output to the message log file because the log service was not activated.

- E2

  Indicates that the message log could not be output to the message log file due to a communication error.

Examples

```
KFCA05201-I SPP1: A service request was received. (E1)
KFCA05410-I SPP1: Updating starts. (E2)
```

If an error other than E1 or E2 is detected, OpenTP1 assigns the message ID number specified in the function dc_logprint() to a message log indicating the error cause. Then, it provides the log to the standard error.

Arguments whose values are set in the UAP

- msgid

  Specify the message ID to be assigned to each message log. The message ID must be in the KFCAn1n2n3n4n5-*x* format (11 characters) and end with a null character. Specify a value from 05000 to 06999 for the serial number (n1n2n3n4n5) output from the UAP.

- pgm_id

  Specify a user-selected value (request source program ID) for identifying the UAP that called the function dc_logprint(). The value must comprise two alphanumeric characters and end with a null character.

- string

  Specify a character string to be output as a message log to the message log file. The character string can be specified with up to 222 characters. The character string must end with a null character.

- info

  Specify NULL.

- color

  Specify the display color of the message log specified in the function dc_logprint () when the message log is output to the NETM operation support terminal. The following colors are available:

  1: Green

  2: Red

  3: White

  4: Blue

  5: Purple

  6: Sky blue

  7: Yellow

  If a value other than the above or a null character is specified, green is assumed to be specified.

- flags

  Specify DCNOFLAGS.

Return values

Return value	Return value (numeric)	Explanation
DC_OK	0	Normal termination.
DCLOGER_PARAM_ARGS	-1900	The value specified for the argument is invalid.
DCLOGER_COMM	-1901	A communication error occurred or the function dc_rpc_open() was not issued.
DCLOGER_MEMORY	-1902	The memory became insufficient.
DCLOGER_DEFFILE	-1904	The system definition is invalid.
DCLOGER_NOT_UP	-1905	The log service is not active.
DCLOGER_HEADER	-1906	An error occurred when the log service acquired the information to be added to the message log.

Note

When a large log is output, return of the function dc_logprint may be delayed. For example, when the volume of output messages greatly increases due to the occurrence of an error, the transaction processing time increases. Note that this may cause a slowdown.

Message exchange processing (dc_mcf_~)

This section gives the syntax and other information of the following functions which are used for communication in message exchange configuration:

- `dc_mcf_adltap`: Delete an application timer start request
- `dc_mcf_ap_info`: Report the application information
- `dc_mcf_ap_info_uoc`: Report the application information to user exit routines
- `dc_mcf_close`: Close the MCF environment
- `dc_mcf_commit`: Commit an MHP
- `dc_mcf_contend`: Terminate continuous-inquiry-response processing
- `dc_mcf_execap`: Activate an application program
- `dc_mcf_mainloop`: Start an MHP service
- `dc_mcf_open`: Open the MCF environment
- `dc_mcf_receive`: Receive a message
- `dc_mcf_recvsync`: Receive a synchronous message#
- `dc_mcf_reply`: Send a response message#
- `dc_mcf_resend`: Resend a message#
- `dc_mcf_rollback`: Enable MHP rollback
- `dc_mcf_send`: Send a message#
- `dc_mcf_sendrecv`: Exchange a synchronous message#
- `dc_mcf_sendsync`: Send a synchronous message#
- `dc_mcf_tactcn`: Establish a connection#
- `dc_mcf_tactle`: Release a logical terminal from shutdown status#
- `dc_mcf_tdctcn`: Release connection#
- `dc_mcf_tdctle`: Shut down a logical terminal#
- `dc_mcf_tdlqle`: Delete a logical terminal's output queue
- `dc_mcf_tempget`: Accept temporary-stored data
- `dc_mcf_tempput`: Update temporary-stored data

- dc_mcf_timer_cancel: Cancel user timer monitoring
- dc_mcf_timer_set: Set user timer monitoring
- dc_mcf_tlscn: Acquire a connection status#
- dc_mcf_tlscom: Acquire the status of MCF communication services
- dc_mcf_tlsle: Acquire a logical terminal status#
- dc_mcf_tlsln: Acquire the acceptance status for a server-type connection establishment request#
- dc_mcf_tofln: Stop accepting server-type connection establishment requests#
- dc_mcf_tonln: Start accepting server-type connection establishment requests#

#: For details, see the applicable *OpenTP1 Protocol* manual.

The functions for message exchange processing (dc_mcf_~) can be used only in UAPs of TP1/Server Base. They cannot be used in UAPs of TP1/LiNK.

dc_mcf_adltap - Delete an application timer start request

Format

- ANSI C, C++

```
#include <dcmcf.h>
int dc_mcf_adltap (DCLONG action, dcmcf_adltapopt *apopt,
                   char *resv01, DCLONG *resv02,
                   char *resv03, char *resv04)
```

- K&R C

```
#include <dcmcf.h>
int dc_mcf_adltap (action, apopt, resv01, resv02, resv03, resv04)
DCLONG             action;
dcmcf_adltapopt    *apopt;
char               *resv01;
DCLONG             *resv02;
char               *resv03;
char               *resv04;
```

Description

The function `dc_mcf_adltap()` deletes a specified application timer start request and cancels startup of the application. Note that this function cannot delete application timer start requests of the `ans` and `cont` types.

Arguments whose values are set in the UAP

- `action`

  Specify `DCMCFAP` to indicate that an application name is to be specified.

- `apopt`

  Set in a `dcmcf_adltapopt` structure the connection information that is to be the subject of this function's processing.

  The following shows the format of the structure:

```
typedef struct {
                 DCLONG    mcfid;          ...Application start
                                                process identifier
                 char      resv01[4];      ...Reserved
                 char      idnam[9];       ...Application name
                 char      resv02[7];      ...Reserved
                 char      resv03[112];    ...Reserved
```

```
char      resv04[376];   ...Reserved
} dcmcf_adltapopt;
```

- mcfid

  Specify the application start process identifier of the application start service that has the target application that is to be processed. The permitted value range is from 1 to 239.

- resv01

  Fill the area with null characters.

- idnam

  Specify the name of the application whose start is to be canceled. The application name must be specified as a maximum of 8 bytes of characters and must end with the null character.

- resv02, resv03, resv04

  Fill the areas with null characters.

■ resv01, resv02, resv03, resv04

Specify NULL.

Return values

Return value	Return value (numeric)	Explanation
DCMCFRTN_00000	0	Normal termination.
DCMCFRTN_71001	-12001	The dc_mcf_adltap() function cannot be accepted because the MCF is under start processing.
DCMCFRTN_71002	-12002	The dc_mcf_adltap() function cannot be accepted because the MCF is under termination processing.
DCMCFRTN_71004	-12004	A memory shortage occurred during dc_mcf_adltap() function processing.
DCMCFRTN_71005	-12005	A communication error occurred. For the cause, see the message log file.
DCMCFRTN_71006	-12006	An internal error occurred. For the cause, see the message log file.
DCMCFRTN_71007	-12007	The specified application name has not been registered.
		No timer start request has been issued for the specified application name.

Return value	Return value (numeric)	Explanation
		The specified application name belongs to an application whose type is inquiry-response or continuous-inquiry-response.
DCMCFRTN_71009	-12009	The dc_mcf_adltap() function is not supported by the applicable application start process.
DCMCFRTN_71010	-12010	Although the request to delete the specified application timer start request was issued, the request was not accepted. For the cause, see the message log file.
DCMCFRTN_72050	-13050	DCMCFAP is not specified in action.
		An unsupported flag is set in action.
DCMCFRTN_72051	-13051	NULL is set in apopt.
DCMCFRTN_72052	-13052	NULL is not set in resv01.
DCMCFRTN_72053	-13053	NULL is not set in resv02.
DCMCFRTN_72054	-13054	NULL is not set in resv03.
DCMCFRTN_72055	-13055	NULL is not set in resv04.
DCMCFRTN_72061	-13061	A value of 0 or smaller or of 240 or greater is specified for mcfid in dcmcf_adltapopt.
DCMCFRTN_72062	-13062	resv01 in dcmcf_adltapopt is not filled with null characters.
DCMCFRTN_72063	-13063	idnam in dcmcf_adltapopt begins with the null character.
DCMCFRTN_72064	-13064	resv02 in dcmcf_adltapopt is not filled with null characters.
DCMCFRTN_72065	-13065	resv03 in dcmcf_adltapopt is not filled with null characters.
DCMCFRTN_72067	-13067	resv04 in dcmcf_adltapopt is not filled with null characters.
DCMCFRTN_72073	-13073	The character string set in idnam in dcmcf_adltapopt is 9 bytes or more in length.
DCMCFRTN_72074	-13074	The character string set in idnam in dcmcf_adltapopt contains an invalid character.

dc_mcf_ap_info - Report the application information

Format

- ANSI C, C++

```
#include <dcmcf.h>
int dc_mcf_ap_info(DCLONG flags, char *mcfid, char *apname,
                   struct DC_MCFAPINFO *apinfo,
                   char *resv01, DCLONG resv02)
```

- K&R C

```
#include <dcmcf.h>
int dc_mcf_ap_info (flags, mcfid, apname, apinfo, resv01,
                   resv02)
DCLONG  flags;
char  *mcfid;
char  *apname;
struct DC_MCFAPINFO  *apinfo;
char  *resv01;
DCLONG  resv02;
```

Description

The function `dc_mcf_ap_info()` acquires various types of application information from an MHP.

This function can be used to report the application information on the MHP that called the function `dc_mcf_ap_info()` or the other MHP. The application information becomes effective only when the function `dc_mcf_ap_info()` is normally terminated.

Argument whose values is set in the UAP

- `flags`

Specify one of the following flags according to the type of the application to be referenced:

`DCMCFMYSELF`

Specify this flag to acquire the application information on the MHP that called function `dc_mcf_ap_info()`.

`DCMCFOTHER`

Specify this flag to acquire the information on a specific application according to the process identifier for MCF communication service in which the application definition is included, and application name.

- mcfid
  - When specifying DCMCFMYSELF for flags

    Specify NULL.
  - When specifying DCMCFOTHER for flags

    Specify a string indicating the MCF communication process identifier or the application startup process identifier in which the definition of the application to be referenced is included.
- apname
  - When specifying DCMCFMYSELF for flags

    Specify NULL.
  - When specifying DCMCFOTHER for flags

    Specify the name of the application to be referenced.

    When specifying an error event name (ERREVT1, ERREVT2, ERREVT3, or ERREVT4), the default value of the application definition, the no-response type DCMCF_NOANS is set in *mcf_ap_type*.
- apinfo

  Specify the address of the area DC_MCFAPINFO which receives the application information.
- resv01

  Specify NULL.
- resv02

  Specify DCNOFLAGS.

Arguments whose values are returned from OpenTP1

- apinfo

  The application information is returned with the structure DC_MCFAPINFO.

  The structure has the following format:

```
struct DC_MCFAPINFO {
                      char mcf_apinfo[4];
                      DCLONG mcf_resv00;
                      char mcf_ap_name[9];
                      char mcf_ap_mcfid[3];
                      char mcf_resv01[4];
                      DCLONG mcf_ap_stat;
                      DCLONG mcf_ap_type;
                      char mcf_sg_name[32];
                      DCLONG mcf_sg_stat;
                      DCLONG mcf_sg_hold;
                      char mcf_sv_name[32];
                      DCLONG mcf_sv_stat;
                      DCLONG mcf_ap_ntmetim;
                      DCLONG mcf_ap_tempsize;
                      DCLONG mcf_ap_msgcnt;
                      DCLONG mcf_ap_trnmode;
                      DCLONG mcf_ap_quekind;
                      char mcf_resv02[72];
                    }
```

- `mcf_apinfo`

  This area is used by the MCF.

- `mcf_resv00`

  This area is used by the MCF.

- `mcf_ap_name`

  The name of the application whose information is to be reported is returned.

- `mcf_ap_mcfid`

  The process identifier for MCF communication service that includes the definition of the application whose information is to be reported is returned.

- `mcf_resv01`

  This area is used by the MCF.

- `mcf_ap_stat`

  The shutdown or release shutdown status of the application is returned with one of the following flags:

  `DCMCF_IN_DACT`: Input shutdown status

  `DCMCF_SC_DACT`: Schedule shutdown status

  `DCMCF_DACTSTAT`: Input and schedule shutdown status

  `DCMCF_ACTSTAT`: Release shutdown status

- `mcf_ap_type`

The type of the application is returned with one of the following flags:

(The type specified in the `type` operand of the `-n` option in the MCF application definition `mcfaalcap` is set here.)

`DCMCF_ANS`: Response type

`DCMCF_NOANS`: Non-response type

`DCMCF_CONT`: Continuous-inquiry-response type

When specifying `DCMCFOTHER` for `flags` and specifying an error event name (`ERREVT1`, `ERREVT2`, `ERREVT3`, or `ERREVT4`) for apname, the actual type is not reported. In this case, the default value of the application definition, no-response type (`DCMCF_NOANS`) is set here.

- `mcf_sg_name`

  The name of the service group corresponding to the application is returned.

- `mcf_sg_stat`

  The shutdown or release shutdown status of the service group is returned with one of the following flags:

  Input shutdown status: `DCMCF_IN_DACT`

  Schedule shutdown status: `DCMCF_SC_DACT`

  Input and schedule shutdown status: `DCMCF_DACTSTAT`

  Release shutdown status: `DCMCF_ACTSTAT`

- `mcf_sg_hold`

  The holding or release holding status of the service group is returned with one of the following flags:

  Input holding status: `DCMCF_IN_HOLD`

  Schedule holding status: `DCMCF_SC_HOLD`

  Input and schedule holding status: `DCMCF_HOLDSTAT`

  Release holding status: `DCMCF_RLSSTAT`

- `mcf_sv_name`

  The name of the service corresponding to the application is returned.

- `mcf_sv_stat`

  The shutdown or release shutdown status of the service is returned with one of the following flags:

  Input shutdown status: `DCMCF_IN_DACT`

Schedule shutdown status: DCMCF_SC_DACT

Input and schedule shutdown status: DCMCF_DACTSTA

Release shutdown status: DCMCF_ACTSTAT

- mcf_ap_ntmetim

  The limit elapsed time for the non-transaction attribute MHP is returned.

  When mcf_ap_trnmode is DCMCF_TRN, 0 is set here.

  (The value specified in the ntmetim operand of the -v option in the MCF application definition mcfaalcap is set here. If the MCF application definition is omitted, the value specified in the ntmetim operand of the -v option in the MCF manager definition mcfmuap is used.)

- mcf_ap_tempsize

  The size of the temporary-stored data storage area for the continuous-inquiry response is returned.

  When mcf_ap_type is not DCMCF_CONT, 0 is set here.

  (The value specified in the tempsize operand of the -n option in the MCF application definition mcfaalcap is set here.)

- mcf_ap_msgcnt

  The maximum number of input messages that can be stored is returned.

  (The value specified in the msgcnt operand of the -n option in the MCF application definition mcfaalcap is set here.)

- mcf_ap_trnmode

  The transaction attribute of the application is returned with the flag as follows.

  (The value specified in the trnmode operand of the -n option in the MCF application definition mcfaalcap is set here.)

  Managed as a transaction: DCMCF_TRN

  Not managed as a transaction: DCMCF_NONTRN

- mcf_ap_quekind

  The queue to which the received message is assigned is returned with the flag as follows.

  (The value specified in the quekind operand of the -g option in the MCF application definition mcfaalcap is set here.)

  When the message is assigned to the disk queue: DCMCF_DISK

  When the message is assigned to the memory queue: DCMCF_MEMORY

- mcf_resv02

  This area is used by the MCF.

Return values

Return value	Return value (numeric)	Explanation
DCMCFRTN_00000	0	Normal termination.
DCMCFRTN_72000	-13000	The function dc_mcf_ap_info() was called from a service other than the MHP service.
DCMCFRTN_72001	-13001	The specified application name is invalid. Combination of the application name and process identifier is invalid.
DCMCFRTN_72016	-13016	The value specified in a parameter is invalid.
Other than the above occurred.		An unprecedented error (e.g., program damage)

Note

When two or more MHPs for ERREVT1, ERREVT2, ERREVT3, or ERREVT4 are started at the same time, the MHPs for the same error event name may have a different application type. For the MHPs other than the MHP that called function dc_mcf_ap_info(), the application type for the error event (ERREVT1, ERREVT2, ERREVT3, or ERREVT4) is not reported. In this case, the default value of the MCF application definition, no-response type is reported.

dc_mcf_ap_info_uoc - Report the application information to user exit routines

Format

- ANSI, C++

```
#include<dcmcf.h>
int dc_mcf_ap_info_uoc(DCLONG flags, char *apname,
                       struct DC_MCFAPINFO_UOC *apinfo)
```

- K&R C

```
#include<dcmcf.h>
int dc_mcf_ap_info_uoc(flags, apname, apinfo)
DCLONG flags;
char *apname;
struct DC_MCFAPINFO_UOC *apinfo;
```

Description

The function `dc_mcf_ap_info_uoc()` returns information about the application specified by the argument `apname` (application information) to the struct `DC_MCFAPINFO_UOC` data area specified by the argument `apinfo`. This application information includes application definitions (application attribute definitions) and application status (status when the function `dc_mcf_ap_info_uoc()` is called). The application information becomes effective only when the function `dc_mcf_ap_info_uoc()` is normally terminated.

This function can only be used to report application information about user applications that can be activated from the communication service on which the user exit routine is running. Furthermore, it does not report the application information about SPPs (application definition `mcfaalcap -g` type=SPP) or system events (application definition `mcfaalcap` *-n* kind=`mcf`).

If a user application that cannot be activated from the communication service on which the user exit routine is running, an SPP, or a system event is specified, it is interpreted as invalid and the function returns with an error, and the return value `DCMCFRTN_72001` is reported.

This function can be used only from user exit routines that edit input messages (user exit routines that determine application names). It cannot be used from user exit routines other than the above. If you attempt to use it from any other user exit routines, system operation is unpredictable.

Arguments whose values are set in the UAP

- flags

  Specify DCNOFLAGS.

- apname

  Specify the name of the application about which you want to acquire the application information.

- apinfo

  Specify the address of the area DC_MCFAPINFO_UOC that receives the application information.

Arguments whose values are returned from OpenTP1

- apinfo

  The application information is returned with the structure DC_MCFAPINFO.

  The structure has the following format:

```
struct  DC_MCFAPINFO_UOC  {
                              char mcf_apinfo[4];
                              DCLONG mcf_resv00;
                              char mcf_ap_name[9];
                              char mcf_ap_mcfid[3];
                              char mcf_resv01[4];
                              DCLONG mcf_ap_stat;
                              DCLONG mcf_ap_type;
                              DCLONG mcf_ap_msgcnt;
                              char mcf_sg_name[32];
                              DCLONG mcf_sg_stat;
                              DCLONG mcf_sg_hold;
                              DCLONG mcf_sg_msgcnt;
                              char mcf_sv_name[32];
                              DCLONG mcf_sv_stat;
                              DCLONG mcf_ap_ntmetim;
                              DCLONG mcf_ap_tempsize;
                              DCLONG mcf_ap_max_msgcnt;
                              DCLONG mcf_ap_trnmode;
                              DCLONG mcf_ap_quekind;
                              char mcf_resv02[64];
                             };
```

  - mcf_apinfo

    This area is used by the MCF.

  - mcf_resv00

    This area is used by the MCF.

  - mcf_ap_name

The name of the application whose information is to be reported is returned.

- mcf_ap_mcfid

  The process identifier for MCF communication service that includes the definition of the application whose information is to be reported is returned.

- mcf_resv01

  This area is used by the MCF.

- mcf_ap_stat

  The shutdown or release shutdown status of the application is returned with one of the following flags:

  Input shutdown status: DCMCF_IN_DACT

  Schedule shutdown status: DCMCF_SC_DACT

  Input and schedule shutdown status: DCMCF_DACTSTAT

  Release shutdown status: DCMCF_ACTSTAT

- mcf_ap_type

  The type of the application is returned with one of the following flags:

  Response type: DCMCF_ANS

  Non-response type: DCMCF_NOANS

  Continuous-inquiry-response type: DCMCF_CONT

  (The type specified in the type operand of the -n option in the MCF application definition mcfaalcap is set here.)

  When specifying DCMCFOTHER for flags and specifying an error event name (ERREVT1, ERREVT2, ERREVT3, or ERREVT4) for apname, the actual type is not reported. In this case, the default value of the application definition, no-response type (DCMCF_NOANS) is set here.

- mcf_ap_msgcnt

  The number of remaining input messages in this application is returned.

- mcf_sg_name

  The name of the service group corresponding to the application is returned.

- mcf_sg_stat

  The shutdown or release shutdown status of the service group is returned with one of the following flags:

  Input shutdown status: DCMCF_IN_DACT

Schedule shutdown status: DCMCF_SC_DACT

Input and schedule shutdown status: DCMCF_DACTSTAT

Release shutdown status: DCMCF_ACTSTAT

- mcf_sg_hold

  The holding or release holding status of the service group is returned with one of the following flags:

  Input holding status: DCMCF_IN_HOLD

  Schedule holding status: DCMCF_SC_HOLD

  Input and schedule holding status: DCMCF_HOLDSTAT

  Release holding status: DCMCF_RLSSTAT

- mcf_sg_msgcnt

  The number of remaining input messages in this service group is returned.

- mcf_sv_name

  The name of the service corresponding to the application is returned.

- mcf_sv_stat

  The shutdown or release shutdown status of the service is returned with one of the following flags:

  Input shutdown status: DCMCF_IN_DACT

  Schedule shutdown status: DCMCF_SC_DACT

  Input and schedule shutdown status: DCMCF_DACTSTAT

  Release shutdown status: DCMCF_ACTSTAT

- mcf_ap_ntmetim

  The limit of time that can be elapsed for the non-transaction attribute MHP is returned. When mcf_ap_trnmode is DCMCF_TRN, 0 is set here.

  (The value specified in the ntmetim operand of the -v option in the MCF application definition mcfaalcap is set here. If the MCF application definition is omitted, the value specified in the ntmetim operand of the -v option in the MCF manager definition mcfmuap is used.)

- mcf_ap_tempsize

  The size of the temporary-stored data storage area for the continuous-inquiry response is returned.

  When mcf_ap_type is not DCMCF_CONT, 0 is set here.

(The value specified in the tempsize operand of the -n option in the MCF application definition mcfaalcap is set here.)

- mcf_ap_max_msgcnt

  The maximum number of input messages that can be stored is returned. (The value specified in the msgcnt operand of the -n option in the MCF application definition mcfaalcap is set here.)

- mcf_ap_trnmode

  The transaction attribute of the application is returned with one of the following flags:

  Managed as a transaction: DCMCF_TRN

  Not managed as a transaction: DCMCF_NONTRN

  (The value specified in the trnmode operand of the -n option in the MCF application definition mcfaalcap is set here.)

- mcf_ap_quekind

  The queue to which the received message is assigned is returned with one of the following flags:

  When the message is assigned to the disk queue: DCMCF_DISK

  When the message is assigned to the memory queue: DCMCF_MEMORY

  (The value specified in the quekind operand of the -g option in the MCF application definition mcfaalcap is set here.)

- mcf_resv02

  This area is used by the MCF.

Return values

Return value	Return value (numeric)	Explanation
DCMCFRTN_00000	0	Normal termination.
DCMCFRTN_72000	-13000	The function dc_mcf_ap_info_uoc() was called from a service other than the MHP service.
DCMCFRTN_72001	-13001	The specified application name is invalid. No information about the specified application could be acquired.
DCMCFRTN_72016	-13016	The value specified in a parameter is invalid.
Other than the above		An unprecedented error (e.g., program damage) occurred.

Note

1. The function `dc_mcf_ap_info_uoc()` can only be used from user exit routines that edit input messages (user exit routines that determine application names), even though no check is performed if this function is called from user exit routines other than the above. If you attempt to use it from any other user exit routines, system operation is unpredictable. For details about user exit routines, see the applicable *OpenTP1 Protocol* manual.
2. This function can only be used to acquire application information about user applications that can be activated from the communication service on which the user exit routine is running. Furthermore, it does not report application information about SPPs (MCF application definition `mcfaalcap -g type`=SPP) or system events (MCF application definition `mcfaalcap -n kind=mcf`).
3. No UAP trace can be acquired.

dc_mcf_close - Close the MCF environment

Format

- ANSI C, C++

```
#include <dcmcf.h>
void  dc_mcf_close(DCLONG flags)
```

- K&R C

```
#include <dcmcf.h>
void  dc_mcf_close (flags)
DCLONG      flags;
```

Description

The function `dc_mcf_close()` closes the environment in which MCF facilities are used. Call the function `dc_mcf_close()` only once in the process before the UAP that called the function `dc_mcf_open()` calls the function `dc_rpc_close()` in the main function.

Argument whose value is set in the UAP

- `flags`

  Specify `DCNOFLAGS`.

Return value

There is no return value of the function `dc_mcf_close()`.

dc_mcf_commit - Commit an MHP

Format

- ANSI C, C++

```
#include <dcmcf.h>
int  dc_mcf_commit(DCLONG action)
```

- K&R C

```
#include <dcmcf.h>
int  dc_mcf_commit (action)
DCLONG      action;
```

Description

The function `dc_mcf_commit()` notifies the UAP at the transaction branch as a root transaction branch making up the transaction, the transaction service, and the resource manager that the global transaction initiated by the MHP has terminated processing normally (the global transaction has been committed).

When the function `dc_mcf_commit()` terminates normally, a new global transaction is generated.

If a global transaction consists of multiple transaction branches [it involves programs other than the MHP which called the function `dc_mcf_commit()`], the entire global transaction will not be committed until each transaction branch is committed. If the global transaction is composed of multiple resource managers, it will not be committed until the results of each resource manager's processing are committed. If the global transaction is not committed, all the transaction branches are rolled back and the function returns with an error, giving the return value `DCMCFRTN_ROLLBACK`.

The function `dc_mcf_commit()` can be called only by an MHP specified as nonresponse-type (`type=noans`) in the MCF application definition. If it is called by an MHP of another type, it returns with an error, giving the return value `DCMCFRTN_72000`. If it is called by a UAP other than an MHP, it also returns with an error, giving the return value `DCMCFRTN_72000`.

Arguments whose value is set in the UAP

- `action`

  Specify `DCNOFLAGS`.

Return values

Return value	Return value (numeric)	Explanation
DCMCFRTN_00000	0	Normal termination. If this return value returns, the process which called the function dc_mcf_commit() has started a new transaction.
DCMCFRTN_ROLLBACK	-11906	The transaction was not committed, but was rolled back. If this return value returns, the process which called the function dc_mcf_commit() has started a new transaction.
DCMCFRTN_HEURISTIC	-11907	The global transaction which called the function dc_mcf_commit() was subjected to a heuristic decision which brought about the following: Some transaction branches were committed, whereas other transaction branches were rolled back. If this return value returns, the process which called the function dc_mcf_commit() has started a new transaction.
DCMCFRTN_HAZARD	-11908	The transaction branch of the global transaction was completed heuristically. However, the synchronization point of the heuristically completed transaction branch cannot be identified. If this return value returns, the process which called the function dc_mcf_commit() has started a new transaction. This function returns DCMCFRTN_HAZARD even when you specify 00000001 for the trn_extend_function operand in the transaction service definition and the return value from the resource manager at one-phase commit is XAER_NOTA.
DCMCFRTN_72000	-13000	If the function returns at MHP execution: The function dc_mcf_commit() was called at a wrong position. The MHP called the function dc_mcf_commit() before the function dc_mcf_receive() for receiving the first segment. The function dc_mcf_commit() was called by an MHP which is not specified as nonresponse-type (type=noans) in the MCF application definition. The function dc_mcf_commit() was called by an MHP with the nontransaction attribute.
		If the function returns at SPP execution: The function dc_mcf_commit() cannot be called by SPPs.
DCMCFRTN_72016	-13016	The value specified for action is invalid.

Return value	Return value (numeric)	Explanation
Other than the above		An unprecedented error (e.g., program damage) occurred.

Notes

Even when the function `dc_mcf_commit()` terminates normally, the input message is not deleted from the input queue. This means that when message processing is restarted after the MHP is rescheduled, the already committed range (up to what point the results of processing have been committed) is unknown. The MHP is rescheduled when:

1. An MCF event is reported to schedule an MHP for MCF event processing.
2. Since the system is terminated abnormally, the OpenTP1 reschedules the MHP for the process.

If message processing is to be continued by the rescheduled MHP, the user is responsible for learning the committed range of processing results.

dc_mcf_contend - Terminate continuous-inquiry-response processing

Format

- ANSI C, C++

```
#include <dcmcf.h>
int  dc_mcf_contend(DCLONG action,char *resv01)
```

- K&R C

```
#include <dcmcf.h>
int  dc_mcf_contend (action, resv01)
DCLONG   action;
char       *resv01;
```

Description

The function `dc_mcf_contend()` terminates continuous-inquiry-response processing. Before terminating continuous-inquiry-response processing, verify that `nextap` of the function `dc_mcf_reply()` called from the MHP is a null character and that the function `dc_mcf_execap()` for activating a cont-type MHP has not been called. If the MHP to be activated next is specified for nextap of the function `dc_mcf_reply()` or if the function `dc_mcf_execap()` for activating a cont-type MHP has been called, the function `dc_mcf_contend()` returns with an error.

After the function `dc_mcf_contend()` is called, the `dc_mcf_tempget()` function and the function `dc_mcf_tempput()` for accessing temporary-stored data cannot be called.

Arguments whose values are set in the UAP

- `action`

  Specify `DCNOFLAGS`.

- `resv01`

  Specify a null character.

Return values

Return value	Return value (numeric)	Explanation
`DCMCFRTM_00000`	0	Normal termination.

Return value	Return value (numeric)	Explanation
DCMCFRTN_72000	-13000	Return at MHP execution The function dc_mcf_contend() was called out of sequence. The function dc_mcf_contend() was called before the function dc_mcf_receive() (for receiving the first segment) was called from the MHP.
		Return at SPP execution The function dc_mcf_contend() cannot be called from an SPP.
DCMCFRTN_72016	-13016	The value specified for action is invalid. The value of the area pointed to by resv01 is not a null character.
DCMCFRTN_72101	-13101	The function dc_mcf_contend() was called from an MHP for which continuous-inquiry-response type (type=cont) was not specified in the MCF application definition.
DCMCFRTN_72107	-13107	The function dc_mcf_contend() was called.
DCMCFRTN_72111	-13111	The continuous-inquiry-response type application to be activated next was specified, a response message was sent (value specified for nextap of the function dc_mcf_reply()), then the function dc_mcf_contend() was called.
		The function dc_mcf_execap() that specified the continuous-inquiry-response type application to be activated next was called, then the function dc_mcf_contend() was called.
Other than the above		An unprecedented error (e.g., program damage) occurred.

dc_mcf_execap - Activate an application program

Format

- ANSI C, C++

```
#include <dcmcf.h>
int  dc_mcf_execap(DCLONG action,DCLONG commform,char *resv01,
                   DCLONG active,char *apnam,char *comdata,
                   DCLONG cdataleng)
```

- K&R C

```
#include <dcmcf.h>
int  dc_mcf_execap (action, commform, resv01, active,
                    apnam, comdata, cdataleng)
DCLONG      action;
DCLONG      commform;
char        *resv01;
DCLONG      active;
char        *apnam;
char        *comdata;
DCLONG      cdataleng;
```

Description

The function `dc_mcf_execap()` starts the MHP or SPP of the application name specified for `apnam` from a UAP (SPP or MHP). After the UAP terminates, it can be started immediately or after a specified interval has passed. After the transaction or service function has terminated, the MHP or SPP with the application name specified for `apnam` can be started immediately or after a preset length of time.

To call the function `dc_mcf_execap()` from an SPP, process the SPP as a transaction and call the function `dc_mcf_open()` in the SPP main function.

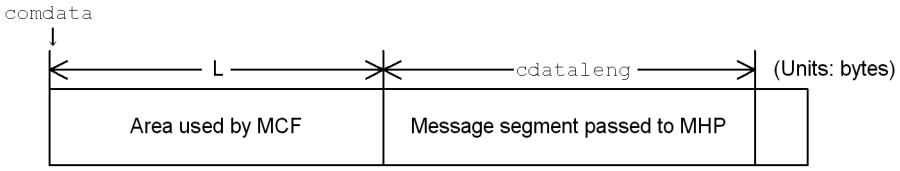
If an MHP is activated by issuing the function `dc_mcf_execap()` from another MHP, the name in the first-received message is used as the logical terminal name of the input source that receives messages through the activated MHP. If the function `dc_mcf_execap()` is called from the MHP, the name in the first-received message is also used as the logical terminal name of the input source that receives messages.

If an MHP is activated by issuing the function `dc_mcf_execap()` from an SPP, an asterisk (`*`) is used as the logical terminal name of the input source that receives messages through the activated MHP. If the function `dc_mcf_execap()` is called from the MHP, an asterisk (`*`) is also used as the logical terminal name of the input source that receives messages.

The maximum length of a single message segment that can be sent is 32 kilobytes. Note that the actual value might be smaller depending on the protocol. For details, see

the applicable *OpenTP1 Protocol* manual.

The figure below shows the segment format of the message to be passed to the MHP to be activated. With buffer format 1, L is 8 bytes; with buffer format 2, L is 4 bytes.



Arguments whose values are set in the UAP

- action

  Specify the following items in the format shown below:

  - Whether the segment to be passed to the MHP or SPP to be activated is the last segment of a logical message
  - When to activate the MHP or SPP
  - Buffer format to be used

```
{DCMCFESI|DCMCFBUF1}[|{DCMCFJUST|DCMCFINTV|DCMCFTIME}]
                                          [|{DCMCFEMI|DCMCFBUF2}]
```

DCMCFESI

Specify DCMCFESI to pass the first segment or an intermediate segment. If the function dc_mcf_execap() with DCMCFESI specified is called, the function dc_mcf_execap() with DCMCFEMI specified for action must be called.

DCMCFEMI

Specify DCMCFEMI to pass the last segment. If the logical message comprises only a single segment, also specify DCMCFEMI. Also specify DCMCFEMI if the sending of the first or an intermediate segment is to be followed by the notice of the completion of message sending.

DCMCFJUST

Specify DCMCFJUST to enable immediate start. The value specified for active is ignored in this case.

DCMCFINTV

Specify DCMCFINTV for an interval timer. The MHP or SPP will be activated the time specified for active after the function dc_mcf_execap() is called.

DCMCFTIME

Specify DCMCFTIME for a time-point timer. The MHP or SPP will be activated at the time specified for active.

DCMCFBUF1

Specify DCMCFBUF1 when using buffer format 1.

DCMCFBUF2

Specify DCMCFBUF2 when using buffer format 2.

- commform

  Specify DCNOFLAGS.

- resv01

  Specify a null character.

- active

  - Interval timer drive (specification of DCMCFINTV for action)

    Specify the number of seconds which will elapse from the call of the function dc_mcf_execap() to the activation of the MHP or SPP. The value must be 1 to 360000 (1 second to 100 hours).

  - Time-point timer drive (specification of DCMCFTIME for action)

    Specify when to activate the MHP or SPP specified for apnam. The time is in seconds relative to 00:00:00 in local time.

    Time setting example

    To activate the MHP or SPP at 2:30:30 p.m. in local time:

```
14*3600+30*60+30=52230
```

    Assign 52230 to active.

  The range of specifiable values is 0 (activation at 00:00:00) to 86399 (activation at 23:59:59).

  The value specified for active is valid only for timer-driven activation. If immediate activation is specified, the value specified for active is ignored.

  Since OpenTP1 checks whether the activation time has been reached at regular intervals, there is a difference between the time specified for active and the actual activation time. The accuracy of time monitoring depends on the value for the time monitoring interval specified for the btim operand in the -t option of the MCF communication configuration definition mcfttim.

- `apnam`

  Specify the application name of the MHP or SPP to be started. The application name can be specified with up to 8 bytes. The application name must end with a null character.

- `comdata`

  Specify the contents of the message segment to be passed to the MHP or SPP which is to start. Specify also segment if the sending of the first or an intermediate segment is to be followed by the notice of the completion of message sending.

- `cdataleng`

  Specify the length of the segment to be passed to the MHP or SPP to be started. Specify 0 for cdataleng if the sending of the first or an intermediate segment is to be followed by the notice of the completion of message sending.

Return values

Return value	Return value (numeric)	Explanation
DCMCFRTN_00000	0	Normal termination.
DCMCFRTN_71002	-12002	An error occurred during input/output processing for the message queue.
		The message queue is in shutdown state.
		No message queue was allocated.
		The value specified for the segment length exceeds 32,000 bytes.
		The MHP or SPP specified for `apnam` cannot be activated because the MCF is being terminated.
DCMCFRTN_71003	-12003	The message queue is full.
DCMCFRTN_71004	-12004	The buffer for storing messages could not be acquired in the memory.
DCMCFRTN_71108	-12108	An attempt was made to start the MHP or SPP of the application name specified for `apnam`, but the MHP's or SPP's management table could not be acquired.
		The local memory of the process is insufficient.
DCMCFRTN_72000	-13000	Return at MHP execution The function `dc_mcf_execap()` was called before the function `dc_mcf_receive()` with `DCMCFFRST` specified for action.

Return value	Return value (numeric)	Explanation
		Return at SPP execution The function `dc_mcf_execap()` is called from a nontransaction SPP process.
`DCMCFRTN_72001`	`-13001`	The specified application name is not defined in the MCF.
		The application name is incorrect.
		The application startup process name is not specified in the communication service definition (`mcfmcname` definition command) for the MCF manager.
		The application startup process identifier is not specified in the MCF application environment definition (the `-p` option of the `mcfaenv` definition command) corresponding to an application startup process.
		The application startup process identifier specified in the application environment definition (the `-p` option of the `mcfaenv` definition command) does not match the identifier specified in the communication configuration definition (the `mcftenv` definition command) for the process.
		For starting of non-response MHPs and SPPs: • No value is specified for the logical terminal (the `lname` operand in the `-n` option of the `mcfaalcap` definition command) in the attribute definition of the application to be started. • The logical terminal specified in the attribute definition of the application to be started is not defined in the communication configuration definition (`mcftalcle` definition command) of the application startup process. • The logical terminal specified in the application attribute definition of the application to be started is not for send-only communication (`mcftalcle -t=send`). • The logical terminal specified in the attribute definition of the application to be started cannot start the application.

Return value	Return value (numeric)	Explanation
		For starting of response and continuous inquiry response MHPs: • The internal communication path (the `cname` operand in the `-n` option of the `mcfaalcap` definition command) is not specified in the attribute definition of the application to be started. • The internal communication path specified in the attribute definition of the application to be started is not defined in the communication configuration definition (the `-c` option of the `mcftpsvr` definition command) of the application startup process. • The inquiry logical terminal (`mcftalcle -t=request`) is not specified in the communication configuration definition (`mcftalcle` definition command) of the application start process.
		When starting an application from an SPP: • The application startup process identifier is not specified in the `mcf_psv_id` operand for the user service or user service default definition of the starting UAP. • The following two values do not match: Application startup process identifier specified in the `mcf_psv_id` operand for the user service or user service default definition of the staring UAP. Application startup process identifier specified in the communication configuration definition (the `-s` option of the `mcftenv` definition command) and application environment definition (the `-p` option of the `mcfaenv` definition command) of the application startup process. • The MCF manager identifier specified in the `mcf_mgrid` operand of the user service or user service default definition of the starting UAP does not match the identifier of the MCF manager to which the application startup process belongs.
`DCMCFRTN_72005`	`-13005`	A value less than 1 byte was specified as the message segment length in the function `dc_mcf_execap()` in which `DCMCFESI` was specified for `action`.
`DCMCFRTN_72007`	`-13007`	From a response type (`type=ans`) MHP which already called the function `dc_mcf_reply()`, another response type MHP was started by the function `dc_mcf_execap()`.

Return value	Return value (numeric)	Explanation
		From a continuous-inquiry-response type (type=cont) MHP which already called the function dc_mcf_reply(), another continuous-inquiry-response type MHP was started by the function dc_mcf_execap().
DCMCFRTN_72009	-13009	From a response type (type=ans) MHP, a response type MHP was started by the function dc_mcf_execap() more than once.
		From a continuous-inquiry-response type (type=cont) MHP, a continuous-inquiry-response type MHP was started by the function dc_mcf_execap() more than once.
DCMCFRTN_72011	-13011	From an MHP which is not response type (type=ans), a response type MHP was started by the function dc_mcf_execap().
		From an MHP which is not continuous-inquiry-response type (type=cont), a continuous-inquiry-response type MHP was started by the function dc_mcf_execap().
DCMCFRTN_72016	-13016	The value specified for action is invalid.
		The value specified for resv01 is not a null character.
		The application start method specified for action is invalid.
		The specified argument is invalid.
DCMCFRTN_72024	-13024	DCNOFLAGS was not specified for commform.
DCMCFRTN_72026	-13026	The value specified as the segment type for action is invalid. DCMCFEMI must be specified for the last segment. DCMCFESI must be specified for a segment other than the last segment.
DCMCFRTN_72041	-13041	The function dc_mcf_execap() with a segment other than the last segment (DCMCFESI) specified was not called for the application name, but the function dc_mcf_execap() with the last segment (DCMCFEMI send segment length = 0) specified was called for the application name.

Return value	Return value (numeric)	Explanation
DCMCFRTN_72044	-13044	From a continuous-inquiry-response type (type=cont) MHP which already called the function dc_mcf_contend(), another continuous-inquiry-response type MHP was started by the function dc_mcf_execap().
DCMCFRTN_72108	-13108	The value specified for active exceeds the limit.
DCMCFRTN_72109	-13109	An attempt was made to activate an MHP, for which type=cont (continuous-inquiry-response type) was specified in the MCF application definition, by the function dc_mcf_execap() with timer start specified.
DCMCFRTN_77001	-18001	The logical terminal (LE) corresponding to the application to be activated is being started and cannot be used, or no logical terminals are available.
Other than the above		An unprecedented error (e.g., program damage) occurred

Note

1. The activation order of application programs varies depending on the mcfmuap -c order specification in the UAP common definition of the MCF manager definition.
2. If you use a single service function to update a TAM or DAM file and call the function dc_mcf_execap() to start an application that will reference the updated file, make sure that the application will lock the file. If the application references the file without locking the file, the data existing before the file was updated might be referenced.

dc_mcf_mainloop - Start an MHP service

Format

■ ANSI C, C++

```
#include <dcmcf.h>
int  dc_mcf_mainloop(DCLONG flags)
```

■ K&R C

```
#include <dcmcf.h>
int  dc_mcf_mainloop (flags)
DCLONG      flags;
```

Description

The function dc_mcf_mainloop() starts accepting service requests to service functions which are included in the service group being executed in the process that called this function. The function dc_mcf_mainloop() does not return until it receives a termination request from OpenTP1.

Argument whose value is set in the UAP

■ flags

Specify DCNOFLAGS.

Return values

Return value	Return value (numeric)	Explanation
DC_OK	0	The function dc_mcf_mainloop() received a termination request from OpenTP1. The UAP that called the function dc_mcf_mainloop() must immediately execute termination processing for its process. Then, the UAP must call the function dc_mcf_close() and the function dc_rpc_close() to enable exit().
DCMCFER_INVALID_ARGS	-11900	The specified argument is invalid.
DCMCFER_PROTO	-11901	The function dc_rpc_open() was not called before the function dc_mcf_mainloop().
DCMCFER_FATAL	-11902	The service could not be started.
DCMCFER_NOMEM	-11903	The memory became insufficient.

dc_mcf_open - Open the MCF environment

Format

- ANSI C, C++

```
#include <dcmcf.h>
int  dc_mcf_open(DCLONG flags)
```

- K&R C

```
#include <dcmcf.h>
int  dc_mcf_open (flags)
DCLONG      flags;
```

Description

The function dc_mcf_open() constructs the environment in which MCF facilities are used. Call the function dc_mcf_open() for UAPs which use MCF facilities.

After the dc_rpc_open() is called, the function dc_mcf_open() must be called in the main function. Issue the function dc_mcf_open() only once in the process before the function dc_mcf_mainloop() (function dc_rpc_mainloop() for an SPP). The following shows when to call the function dc_mcf_open():

```
dc_rpc_open()
dc_mcf_open()
dc_mcf_mainloop() (dc_rpc_mainloop() for an SPP)
   :
   :
dc_mcf_close()
dc_rpc_close()
```

Argument whose value is set in the UAP

- flags

  Specify DCNOFLAGS.

Return values

Return value	Return value (numeric)	Explanation
DC_OK	0	Normal termination.
DCMCFER_INVALID_ARGS	-11900	The value specified for flags is invalid.
DCMCFER_PROTO	-11901	The function dc_rpc_open() was not called.

Return value	Return value (numeric)	Explanation
		The function `dc_mcf_open()` was called.
`DCMCFER_FATAL`	`-11902`	Initialization processing was unsuccessful.
`DCMCFER_NOMEM`	`-11903`	The memory became insufficient.

dc_mcf_receive - Receive a message

Format

- ANSI C, C++

```
#include <dcmcf.h>
int  dc_mcf_receive(DCLONG action,DCLONG commform,char *termnam,
                    char *resv01,char *recvdata,DCLONG *rdataleng,
                    DCLONG inbufleng,DCLONG *time)
```

- K&R C

```
#include <dcmcf.h>
int  dc_mcf_receive (action, commform, termnam, resv01,
                     recvdata, rdataleng, inbufleng, time)
DCLONG    action;
DCLONG    commform;
char    *termnam;
char    *resv01;
char    *recvdata;
DCLONG    *rdataleng;
DCLONG    inbufleng;
DCLONG    *time;
```

Description

The function `dc_mcf_receive()` receives a segment of a message. When a whole logical message is received, call this function as many times as there are segments.

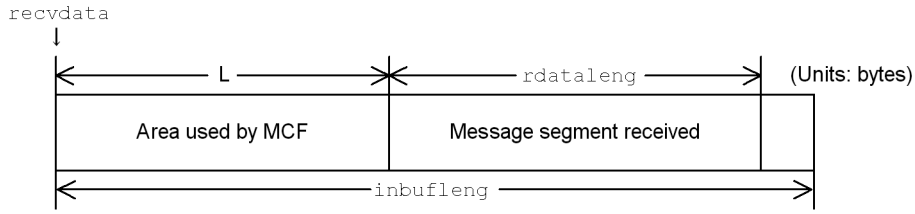
The function `dc_mcf_receive()` can receive the following messages:

- Messages which are sent from the remote system via communication protocol
- MCF events which are reported from the local system
- Messages which are sent by the function `dc_mcf_execap()` (Activate an application program) from a UAP of the local system
- Messages which are sent by executing the `mcfuevt` command on the local system

When receiving a message which is sent from the remote system via communication protocol, the syntax of the function `dc_mcf_receive()` varies according to communication protocol in use. For the syntax of the function `dc_mcf_receive()` which receives a message from the remote system, see the explanation in the applicable *OpenTP1 Protocol* manual.

The maximum length of a single segment that can be received is 1 megabyte. Note that the actual value might be smaller depending on the protocol. For details, see the applicable *OpenTP1 Protocol* manual.

The figure below shows the format of the receive segment area. With buffer format 1, L is 8 bytes; with buffer format 2, L is 4 bytes.



Arguments whose values are set in the UAP

- action

  Specify whether the first segment of the message is received and the buffer format to be used in the format shown below:

```
{DCMCFFRST|DCMCFSEG}[|{DCMCFBUF1|DCMCFBUF2}]
```

  DCMCFFRST

  Specify DCMCFFRST to receive the first segment. If the message comprises only a single segment, also specify DCMCFFRST.

  DCMCFSEG

  Specify DCMCFSEG to receive an intermediate segment or the last segment.

  DCMCFBUF1

  Specify DCMCFBUF1 when using buffer format 1. In general, buffer format 1 is used.

  DCMCFBUF2

  Specify DCMCFBUF2 when using buffer format 2.

- commform

  Specify DCNOFLAGS.

- termnam [when an intermediate segment or the last segment is received]

  Specify the input logical terminal name. Specify the logical terminal name returned when the first segment is received.

- resv01

  Specify a null character.

- `recvdata`

  Specify the receive segment area. When the message is sent from the local system, the maximum length of receive segment is 32,000 bytes.

  When the message is sent from the remote system, the maximum length of receive segment depends on the product adopting the communication protocol.

  When the function `dc_mcf_receive()` terminates, a segment of the message is returned.

- `inbufleng`

  Specify the length of the receive segment area.

Arguments whose values are returned from OpenTP1

- `termnam` [when the first segment is received]

  The input logical terminal name is returned.

  Specify the returned logical terminal name when an intermediate segment or the last segment is received.

- `recvdata`

  The contents of the receive segment are returned.

- `rdataleng`

  The length of the receive segment is returned.

- `time`

  The time when the message is received is returned in total seconds since 00:00:00 on January 1, 1970.

Return values

Return values	Return value (numeric)	Explanation
`DCMCFRTN_00000`	`0`	Normal termination.
`DCMCFRTN_71000`	`-12000`	The function `dc_mcf_receive()` for receiving the first segment was called more than once. To receive an intermediate segment or the last segment, call the function `dc_mcf_receive()` with `DCMCFSEG` specified for action.

Return values	Return value (numeric)	Explanation
DCMCFRTN_71001	-12001	The function dc_mcf_receive() for receiving the next segment was called after the last segment of the message is received. The function dc_mcf_receive() called immediately before receives a message completely. If the function dc_mcf_receive() is called again after this value is returned, the return value DCMCFRTN_72000 is returned.
DCMCFRTN_71002	-12002	An error occurred during input processing for the message queue.
		The message queue is in shutdown state.
DCMCFRTN_72000	-13000	Return at MHP execution The function dc_mcf_receive() for receiving an intermediate segment or the last segment was called before the function dc_mcf_receive() for receiving the first segment was called. To receive the first segment, call the function dc_mcf_receive() with DCMCFFRST specified for action. The function dc_mcf_receive() was called again after the return value DCMCFRTN_71001 was returned.
		Return at SPP execution The function dc_mcf_receive() cannot be called from an SPP.
DCMCFRTN_72001	-13001	The logical terminal name specified for termnam is invalid.
DCMCFRTN_72013	-13013	A segment exceeding the length of the receive area was received. The excess portion was truncated.
DCMCFRTN_72016	-13016	The value specified for action is invalid.
		The value specified for resv01 is invalid.
		The value specified for the argument is invalid.
DCMCFRTN_72024	-13024	The value specified for commform is invalid.
DCMCFRTN_72025	-13025	The value of the segment type specified for action is invalid. The value must be DCMCFFRST or DCMCFSEG.
DCMCFRTN_72036	-13036	The segment receive area is insufficient. Allocate an area of 9 bytes or more for buffer format 1; 5 bytes or more for buffer format 2.

Return values	Return value (numeric)	Explanation
Other than the above		An unprecedented error (e.g., program damage) occurred.

dc_mcf_recvsync - Receive a synchronous message

Format

For details of the format, see the explanation in the applicable *OpenTP1 Protocol* manual.

Description

The function `dc_mcf_recvsync()` receives a logical message from other system during the processing of an active UAP. When the function `dc_mcf_recvsync()` is called by a UAP, it searches the input queue for a message sent from the logical terminal name specified in it and receives the message. If there is not such a message, the function waits until an appropriate message arrives. In this way, the reception of a logical message is synchronized with the call of the function `dc_mcf_recvsync()` from the UAP.

The function receives a segment of a logical message. If the logical message consists of one segment, the function `dc_mcf_recvsync()` must be issued only once. If the logical message consists of multiple segments, the function `dc_mcf_recvsync()` must be called as many times as the segments to receive the logical message.

The maximum length of a single segment that can be received is 1 megabyte. Note that the actual value might be smaller depending on the protocol. For details, see the applicable *OpenTP1 Protocol* manual.

The MCF area which holds the segment received by the function `dc_mcf_recvsync()` consists of the area used by the MCF and the area actually holding the received message segment.

The values to be specified for the arguments and the return values vary with the communication protocol in use. For details, see the applicable *OpenTP1 Protocol* manual.

dc_mcf_reply - Send a response message

Format

For details on the format, see the explanation in the applicable *OpenTP1 Protocol* manual.

Description

The function `dc_mcf_reply()` sends a logical message in response to other system. It sends a response to the logical terminal from which a message was received by the function `dc_mcf_receive()`.

The function `dc_mcf_reply()` can be called only by MHPs whose application type is `ans` or `cont`.

The function sends a segment of a logical message as a response. If the received logical message consists of one segment, the function `dc_mcf_reply()` must be called only once to send a response. If the received logical message consists of multiple segments, the function `dc_mcf_reply()` must be called as many times as the segments to send one logical message in response.

The application which is under MCF control (MHP service function) allows the MCF to send a message after the function `dc_mcf_reply()` is issued to send the logical message to its end and the MHP terminates normally. In this way, message sending by the function `dc_mcf_reply()` is asynchronous with MHP processing.

The maximum length of a single message segment that can be sent is 32 kilobytes. Note that the actual value might be smaller depending on the protocol. For details, see the applicable *OpenTP1 Protocol* manual.

The MCF area which holds the segment to be sent by the function `dc_mcf_reply()` consists of the area used by the MCF and the area actually holding the message segment to be transmitted in response.

The values to be specified for the arguments and the return values vary with the communication protocol in use. For details, see the applicable *OpenTP1 Protocol* manual.

dc_mcf_resend - Resend a message

Format

For details on the format, see the explanation in the applicable *OpenTP1 Protocol* manual.

Description

The function `dc_mcf_resend()` resends an already sent logical message to other system. The resent message is treated as a new message separate from the already sent message. The message to be resent can be selected using information about already sent messages as follows:

- Output-destination logical terminal name
- Message sequence number
- Message type (general branch or priority branch)

Before a node can use the function `dc_mcf_resend()`, it must use a queue (disk queue) for holding already sent messages.

If the message to be resent was not sent, the function `dc_mcf_resend()` returns with an error. It also returns with an error if the message to be resent is not found in the output queue.

The values to be specified for the arguments and the return values vary with the communication protocol in use. For details, see the explanation in the applicable *OpenTP1 Protocol* manual.

Note

The message resend order varies depending on the `mcfmuap -c order` specification in the UAP common definition of the MCF manager definition.

dc_mcf_rollback - Enable MHP rollback

Format

- ANSI C, C++

```
#include <dcmcf.h>
int  dc_mcf_rollback(DCLONG action)
```

- K&R C

```
#include <dcmcf.h>
int  dc_mcf_rollback (action)
DCLONG      action;
```

Description

The function dc_mcf_rollback() cancels processing between when the MHP service program that defines the transaction attribute is started and when the function dc_mcf_rollback() is called. If DCMCFRTRY is specified for action, processing between when the MHP is started and when the function dc_mcf_rollback() is called is canceled, and the canceled MHP processing is rescheduled.

Arguments whose values are set in the UAP

- action

  Specify DCMCFRTRY, DCMCFRRTN, or DCMCFNRTN for the type of rollback.

  DCMCFRTRY

  Processing between the MHP is started and when the function dc_mcf_rollback() is called is canceled, and the canceled MHP processing is rescheduled (any received messages are stored at the end of the relevant input queue and the MHP is rescheduled). Control does not return from the function dc_mcf_rollback(), and the process is terminated.

  DCMCFRRTN

  Processing between the MHP is started and when the function dc_mcf_rollback() is called is canceled, and control returns. Processing after the normal termination of the function dc_mcf_rollback() with DCMCFRRTN specified is treated as another transaction.

  DCMCFNRTN

  Processing between the MHP is started and when the function dc_mcf_rollback() is called is canceled. Control does not return from the function dc_mcf_rollback(), and the process is terminated.

Return values

Return value	Return value (numeric)	Explanation
DCMCFRTN_00000	0	Normal termination.
DCMCFRTN_72000	-13000	Return at MHP execution The function dc_mcf_rollback() was called out of sequence. The function dc_mcf_rollback() with DCMCFRRTN specified for action was called before the function dc_mcf_receive() (for receiving the first segment) was called from the MHP.The function dc_mcf_rollback() was called by an MHP with the nontransaction attribute.
		Return at SPP execution The function dc_mcf_rollback() cannot be called from an SPP.
DCMCFRTN_72027	-13027	The value specified for action is invalid.
Other than the above		An unprecedented error (e.g., program damage) occurred.

dc_mcf_send - Send a message

Format

For details on the format, see the explanation in the applicable *OpenTP1 Protocol* manual.

Description

The function `dc_mcf_send()` sends a logical message to other system.

The function sends a segment of a logical message. If the sent logical message consists of one segment, the function `dc_mcf_send()` must be called only once. If the sent logical message consists of multiple segments, the function `dc_mcf_send()` must be called as many times as the segments to send one logical message.

The application which is under MCF control (MHP service function) or SPP allows the function `dc_mcf_send()` to send messages asynchronously to UAP processing.

The maximum length of a single message segment that can be sent is 32 kilobytes. Note that the actual value might be smaller depending on the protocol. For details, see the applicable *OpenTP1 Protocol* manual.

The MCF area which holds the segment to be sent by the function `dc_mcf_send()` consists of the area used by the MCF and the area actually holding the message segment to be sent.

The values to be specified for the arguments and the return values vary with the communication protocol in use. For details, see the applicable *OpenTP1 Protocol* manual.

Note

The message send order varies depending on the `mcfmuap -c order` specification in the UAP common definition of the MCF manager definition.

dc_mcf_sendrecv - Exchange a synchronous message

Format

For details on the format, see the explanation in the applicable *OpenTP1 Protocol* manual.

Description

The function `dc_mcf_sendrecv()` sends a logical message to other system, during the processing of an active UAP and receives a response from the logical terminal. Once the function `dc_mcf_sendrecv()` is called by a UAP, it waits until message sending to the logical terminal designated in the function and response arrival are completed. In this way, the sending and reception of a logical message is synchronized with the call of the function `dc_mcf_sendrecv()` from the UAP.

The function `dc_mcf_sendrecv()` enters the state of wait for a response when the MCF sends a message by making the function `dc_mcf_sendrecv()` send the last segment of the message.

The function `dc_mcf_sendrecv()` sends a segment of a logical message. If the logical message consists of one segment, the function `dc_mcf_sendrecv()` must be called only once. If the logical message consists of multiple segments, the function `dc_mcf_sendrecv()` must be called as many times as the segments to send the logical message.

When the MCF receives all segments of the response message from the logical terminal, the function `dc_mcf_sendrecv()` that sent the last segment receives only the first segment of the response message. The intermediate and subsequent segments are received by the function `dc_mcf_recvsync()`.

The maximum length of a single segment that can be received is 1 megabyte. Note that the actual value might be smaller depending on the protocol. The maximum length of a single message segment that can be sent is 32 kilobytes. Note that the actual value might be smaller depending on the protocol. For details, see the applicable *OpenTP1 Protocol* manual.

The MCF area which holds the segment to be sent by the function `dc_mcf_sendrecv()` consists of the area used by the MCF and the area actually holding the message segment to be sent.

The values to be specified for the arguments and the return values vary with the communication protocol in use. For details, see the applicable *OpenTP1 Protocol* manual.

dc_mcf_sendsync - Send a synchronous message

Format

For details on the format, see the explanation in the applicable *OpenTP1 Protocol* manual.

Description

The function `dc_mcf_sendsync()` sends a logical message to other system, during the processing of an active UAP. Once the function `dc_mcf_sendsync()` is called by a UAP, it waits until the message is written in the output queue and is completely sent to the logical terminal designated in the function. In this way, the sending of a logical message is synchronized with the call of the function `dc_mcf_sendsync()` from the UAP.

The function sends a segment of a logical message. If the logical message consists of one segment, the function `dc_mcf_sendsync()` must be called only once. If the logical message consists of multiple segments, the function `dc_mcf_sendsync()` must be called as many times as the segments to send the logical message.

The maximum length of a single message segment that can be sent is 32 kilobytes. Note that the actual value might be smaller depending on the protocol. For details, see the applicable *OpenTP1 Protocol* manual.

The MCF area which holds the segment to be sent by the function `dc_mcf_sendsync()` consists of the area used by the MCF and the area actually holding the message segment to be sent.

The values to be specified for the arguments and the return values vary with the communication protocol in use. For details, see the applicable *OpenTP1 Protocol* manual.

dc_mcf_tactcn - Establish a connection

Format

- ANSI C, C++

```
#include <dcmcf.h>
int dc_mcf_tactcn (DCLONG action, dcmcf_tactcnopt *cnopt,
                   char *proinf, DCLONG *resv02, char *resv03,
                   char *resv04)
```

- K&R C

```
#include <dcmcf.h>
int dc_mcf_tactcn (action, cnopt, proinf, resv02, resv03, resv04)
DCLONG             action;
dcmcf_tactcnopt    *cnopt;
char               *proinf;
DCLONG             *resv02;
char               *resv03;
char               *resv04;
```

Description

The `dc_mcf_tactcn()` function establishes a connection.

Normal termination of the `dc_mcf_tactcn()` function indicates that the connection establishment request was accepted successfully by the protocol product. However, this does not necessarily mean that connection with the remote system has been established.

If you intend to perform any connection-related operation after calling the `dc_mcf_tactcn()` function, first use the `dc_mcf_tlscn()` function to check the connection status.

Arguments whose values are set in the UAP

- `action`

Depending on the communication protocol, specify in one of the following formats the method used to specify for this function the connection that is established:

{DCMCFLE|DCMCFCN}[|DCMCFPRO]

DCMCFLE

Specifies that a logical terminal name is specified for the connection that is

established.

This argument is not supported by TP1/NET/NCSB or TP1/NET/X25-Extended.

`DCMCFCN`

Specifies that a connection ID is specified for the connection that is established.

`DCMCFPRO`

Specifies that the function depends on the communication protocol being used.

- `cnopt`

Set in a `dcmcf_tactcnopt` structure the connection information that is to be subject to this function's processing.

The following shows the format of the structure:

```
typedef struct {
                DCLONG        mcfid;      ...MCF communication
                                              process identifier
                char          resv01[4];  ...Reserved
                char          idnam[9];   ...Logical terminal name
                                              or connection ID
                char          resv02[7];      ...Reserved
                char          resv03[112];    ...Reserved
                char          scnnam[9];      ...Area used by MCF
                char          resv04[7];      ...Reserved
                char          yournam[9];     ...Area used by MCF
                char          resv05[7];      ...Reserved
                char          hostnam[143];   ...Area used by MCF
                char          resv06[17];     ...Reserved
                char          resv07[184];    ...Reserved
                } dcmcf_tactcnopt;
```

  - `mcfid`

    Specify the MCF communication process identifier of the MCF communication service for the connection to be processed. The permitted value range is from `0` to `239`.

    This argument is ignored when a logical terminal name is used to request connection establishment.

    If you specify `0`, the system searches for the MCF communication service to which the specified connection ID belongs. In a configuration where many MCF communication services are running or when you issue this function many times from a UAP, we recommend that you specify the MCF communication process identifier.

- resv01

  Fill the area with null characters.

- idnam

  Specify the logical terminal name or connection ID of the connection to be established. The logical terminal name or connection ID must be specified as a maximum of 8 bytes of characters and must end with the null character.

- resv02,resv03, scnnam, resv04, yournam, resv05, hostnam, resv06, resv07

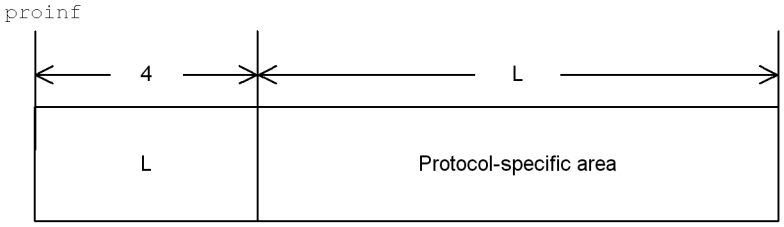
  Fill the areas with null characters.

■ proinf

Specify a protocol-specific area.

If you do not use a function that depends on the communication protocol, specify NULL.

The following shows the format of a protocol-specific area:



The maximum size of a protocol-specific area is 1024 bytes.

The permitted value depends on the communication protocol being used. For details, see the applicable *OpenTP1 Protocol* manual.

■ resv02,resv03, resv04

Specify NULL.

Return values

Return value	Return value (numeric)	Explanation
DCMCFRTN_00000	0	Normal termination.
DCMCFRTN_71001	-12001	The dc_mcf_tactcn() function cannot be accepted because the MCF is under start processing.

Return value	Return value (numeric)	Explanation
DCMCFRTN_71002	-12002	The dc_mcf_tactcn() function cannot be accepted because the MCF is under termination processing.
DCMCFRTN_71004	-12004	A memory shortage occurred during dc_mcf_tactcn() function processing.
DCMCFRTN_71005	-12005	A communication error occurred. For the cause, see the message log file.
DCMCFRTN_71006	-12006	An internal error occurred. For the cause, see the message log file.
DCMCFRTN_71007	-12007	The specified connection name has not been registered.
DCMCFRTN_71008	-12008	The specified logical terminal name has not been registered.
DCMCFRTN_71009	-12009	The dc_mcf_tactcn() function is not supported by the applicable MCF communication process.
DCMCFRTN_71010	-12010	Although the request to establish a connection was issued to the MCF communication process, the request was not accepted. For the cause, see the message log file.
DCMCFRTN_71011	-12011	The dc_mcf_tactcn() function cannot be accepted because the connection has been deleted.
DCMCFRTN_71014	-12014	The specified logical terminal name belongs to TP1/NET/NCSB or TP1/NET/X25-Extended; or, the specified connection group name belongs to TP1/NET/OSI-TP or TP1/NET/TCP/IP.
DCMCFRTN_72050	-13050	An unsupported flag is set in action.
DCMCFRTN_72051	-13051	NULL is set in cnopt.
DCMCFRTN_72052	-13052	When DCMCFPRO is not set in action: NULL is not set in proinf.
		When DCMCFPRO is set in action: A value smaller than 0 or a value 1025 or greater is specified for the size of protocol-specific area L pointed to by proinf.
DCMCFRTN_72053	-13053	NULL is not set in resv02.
DCMCFRTN_72054	-13054	NULL is not set in resv03.
DCMCFRTN_72055	-13055	NULL is not set in resv04.
DCMCFRTN_72060	-13060	DCMCFLE and DCMCFCN cannot be specified together in action.
DCMCFRTN_72061	-13061	A value smaller than 0 or a value 240 or greater is specified for mcfid in dcmcf_tactcnopt.

Return value	Return value (numeric)	Explanation
DCMCFRTN_72062	-13062	resv01 in dcmcf_tactcnopt is not filled with null characters.
DCMCFRTN_72063	-13063	idnam in dcmcf_tactcnopt begins with the null character.
DCMCFRTN_72064	-13064	resv02 in dcmcf_tactcnopt is not filled with null characters.
DCMCFRTN_72065	-13065	resv03 in dcmcf_tactcnopt is not filled with null characters.
DCMCFRTN_72066	-13066	scnnam in dcmcf_tactcnopt is not filled with null characters.
DCMCFRTN_72067	-13067	resv04 in dcmcf_tactcnopt is not filled with null characters.
DCMCFRTN_72068	-13068	yournam in dcmcf_tactcnopt is not filled with null characters.
DCMCFRTN_72069	-13069	resv05 in dcmcf_tactcnopt is not filled with null characters.
DCMCFRTN_72070	-13070	hostnam in dcmcf_tactcnopt is not filled with null characters.
DCMCFRTN_72071	-13071	resv06 in dcmcf_tactcnopt is not filled with null characters.
DCMCFRTN_72072	-13072	resv07 in dcmcf_tactcnopt is not filled with null characters.
DCMCFRTN_72073	-13073	The character string set in idnam in dcmcf_tactcnopt is 9 or more bytes in length.
DCMCFRTN_72074	-13074	The character string set in idnam in dcmcf_tactcnopt contains an invalid character.

dc_mcf_tactle - Release a logical terminal from shutdown status

Format

- ANSI C, C++

```
#include <dcmcf.h>
int dc_mcf_tactle (DCLONG action, dcmcf_tactleopt *leopt,
                   char *proinf, DCLONG *resv02,
                   char *resv03, char *resv04)
```

- K&R C

```
#include <dcmcf.h>
int dc_mcf_tactle (action, leopt, proinf, resv02, resv03, resv04)
DCLONG             action;
dcmcf_tactleopt    *leopt;
char               *proinf;
DCLONG             *resv02;
char               *resv03;
char               *resv04;
```

Description

The `dc_mcf_tactle()` function releases a logical terminal from shutdown status.

Normal termination of the `dc_mcf_tactle()` function indicates that the logical terminal shutdown release request was accepted successfully by the protocol product. However, this does not necessarily mean that the logical terminal has been released from shutdown status.

If you intend to perform any operation related to the logical terminal after calling the `dc_mcf_tactle()` function, first use the `dc_mcf_tlsle()` function to check the logical terminal's status.

Arguments whose values are set in the UAP

- `action`

  Depending on the communication protocol, specify in one of the following formats the method used to specify for this function the logical terminal that is released from shutdown status:

  `DCMCFLE[|DCMCFPRO]`

  `DCMCFLE`

Specifies that the logical terminal name is used.

`DCMCFPRO`

Specifies that the function depends on the communication protocol being used.

- `leopt`

Set in a `dcmcf_tactleopt` structure the information about the logical terminal that is to be the subject of this function's processing.

The following shows the format of the structure:

```
typedef struct {
                DCLONG    mcfid;         ...MCF communication
                                             process identifier
                char      resv01[4];     ...Reserved
                char      idnam[9];      ...Logical terminal name
                char      resv02[7];     ...Reserved
                char      resv03[112];   ...Reserved
                char      resv04[376];   ...Reserved
                } dcmcf_tactleopt;
```

- `mcfid`

  Specify the MCF communication process identifier of the MCF communication service for the logical terminal to be processed. The permitted value range is from `0` to `239`.

  If you specify `0`, the system searches for the MCF communication service to which the specified logical terminal name belongs. In a configuration where many MCF communication services are running or when you issue this function many times from a UAP, we recommend that you specify the MCF communication process identifier.

- `resv01`

  Fill the area with null characters.

- `idnam`

  Specify the name of the logical terminal that is released from shutdown status. The logical terminal name must be specified as a maximum of 8 bytes of characters and must end with the null character.

- `resv02`, `resv03`, `resv04`

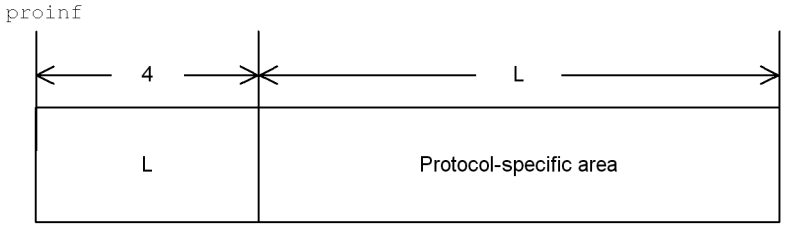
  Fill the areas with null characters.

- `proinf`

Specify a protocol-specific area.

If you do not use a function that depends on the communication protocol, specify NULL.

The following shows the format of a protocol-specific area.



The maximum size of a protocol-specific area is 1024 bytes.

The permitted value depends on the communication protocol being used. For details, see the applicable *OpenTP1 Protocol* manual.

- resv02, resv03, resv04

  Specify NULL.

Return values

Return value	Return value (numeric)	Explanation
DCMCFRTN_00000	0	Normal termination.
DCMCFRTN_71001	-12001	The dc_mcf_tactle() function cannot be accepted because the MCF is under start processing.
DCMCFRTN_71002	-12002	The dc_mcf_tactle() function cannot be accepted because the MCF is under termination processing.
DCMCFRTN_71004	-12004	A memory shortage occurred during dc_mcf_tactle() function processing.
DCMCFRTN_71005	-12005	A communication error occurred. For the cause, see the message log file.
DCMCFRTN_71006	-12006	An internal error occurred. For the cause, see the message log file.
DCMCFRTN_71008	-12008	The specified logical terminal name has not been registered.
DCMCFRTN_71009	-12009	The dc_mcf_tactle() function is not supported by the applicable MCF communication process.
DCMCFRTN_71010	-12010	Although the request to release the logical terminal from shutdown status was issued to the MCF communication process, the request was not accepted. For the cause, see the message log file.

Return value	Return value (numeric)	Explanation
DCMCFRTN_71011	-12011	The dc_mcf_tactle() function cannot be accepted because the logical terminal has been deleted.
DCMCFRTN_72050	-13050	DCMCFLE is not set in action.
		An unsupported flag is set in action.
DCMCFRTN_72051	-13051	NULL is set in leopt.
DCMCFRTN_72052	-13052	When DCMCFPRO is not set in action: NULL is not set in proinf.
		When DCMCFPRO is set in action: A value smaller than 0 or a value 1025 or greater is specified for the size of protocol-specific area L pointed to by proinf.
DCMCFRTN_72053	-13053	NULL is not set in resv02.
DCMCFRTN_72054	-13054	NULL is not set in resv03.
DCMCFRTN_72055	-13055	NULL is not set in resv04.
DCMCFRTN_72061	-13061	A value smaller than 0 or a value 240 or greater is specified for mcfid in dcmcf_tactleopt.
DCMCFRTN_72062	-13062	resv01 in dcmcf_tactleopt is not filled with null characters.
DCMCFRTN_72063	-13063	idnam in dcmcf_tactleopt begins with the null character.
DCMCFRTN_72064	-13064	resv02 in dcmcf_tactleopt is not filled with null characters.
DCMCFRTN_72065	-13065	resv03 in dcmcf_tactleopt is not filled with null characters.
DCMCFRTN_72067	-13067	resv04 in dcmcf_tactleopt is not filled with null characters.
DCMCFRTN_72073	-13073	The character string set in idnam in dcmcf_tactleopt is 9 or more bytes in length.
DCMCFRTN_72074	-13074	The character string set in idnam in dcmcf_tactleopt contains an invalid character.

dc_mcf_tdctcn - Release a connection

Format

- ANSI C, C++

```
#include <dcmcf.h>
int dc_mcf_tdctcn (DCLONG action, dcmcf_tdctcnopt *cnopt,
                   char *proinf, DCLONG *resv02, char *resv03,
                   char *resv04)
```

- K&R C

```
#include <dcmcf.h>
int dc_mcf_tdctcn (action, cnopt, proinf, resv02, resv03, resv04)
DCLONG               action;
dcmcf_tdctcnopt      *cnopt;
char                 *proinf;
DCLONG               *resv02;
char                 *resv03;
char                 *resv04;
```

Description

The `dc_mcf_tdctcn()` function releases a connection.

Normal termination of the `dc_mcf_tdctcn()` function indicates that the connection release request was accepted successfully by the protocol product. However, this does not necessarily mean that the connection with the remote system has been released.

If you intend to perform any connection-related operation after calling the `dc_mcf_tdctcn()` function, first use the `dc_mcf_tlscn()` function to check the status of the connection.

Arguments whose values are set in the UAP

- `action`

Depending on the communication protocol, specify in one of the following formats the method used to specify for this function the connection that is released:

{DCMCFLE|DCMCFCN}[|DCMCFFRC][|DCMCFPRO]

DCMCFLE

Specifies that a logical terminal name is specified for the connection that is released.

This argument is not supported by TP1/NET/NCSB or TP1/NET/X25-Extended.

`DCMCFCN`

Specifies that a connection ID is specified for the connection that is released.

`DCMCFFRC`

Specifies that a connection is released forcibly.

`DCMCFPRO`

Specifies that the function depends on the communication protocol being used.

- `cnopt`

Set in the `dcmcf_tdctcnopt` structure the connection information to be the subject of this function's processing.

The following shows the format of the structure:

```
typedef struct {
                DCLONG    mcfid;         ...MCF communication
                                             process identifier
                char      resv01[4];     ...Reserved
                char      idnam[9];      ...Logical terminal name
                                             or connection ID
                char      resv02[7];     ...Reserved
                char      resv03[112];   ...Reserved
                char      scnnam[9];     ...Area used by MCF
                char      resv04[7];     ...Reserved
                char      resv05[360];   ...Reserved
                } dcmcf_tdctcnopt;
```

- `mcfid`

  Specify the MCF communication process identifier of the MCF communication service for the connection to be processed. The permitted value range is from `0` to `239`.

  This argument is ignored when a logical terminal name is used to request a connection release.

  If you specify `0`, the system searches for the MCF communication service to which the specified connection ID belongs. In a configuration where many MCF communication services are running or when you issue this function many times from a UAP, we recommend that you specify the MCF communication process identifier.

- `resv01`

Fill the area with null characters.

- `idnam`

  Specify the logical terminal name or connection ID of the connection to be released. The logical terminal name or connection ID must be specified as a maximum of 8 bytes of characters and must end with the null character.

- `resv02`, `resv03`, `scnnam`, `resv04`, `resv05`

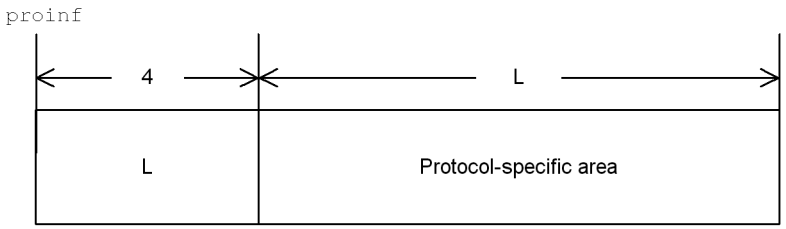
  Fill the areas with null characters.

■ `proinf`

Specify a protocol-specific area.

If you do not use a function that depends on the communication protocol, specify `NULL`.

The following shows the format of a protocol-specific area:



The maximum size of a protocol-specific area is 1024 bytes.

The permitted value depends on the communication protocol being used. For details, see the applicable *OpenTP1 Protocol* manual.

■ `resv02`, `resv03`, `resv04`

Specify `NULL`.

Return values

Return value	Return value (numeric)	Explanation
DCMCFRTN_00000	0	Normal termination.
DCMCFRTN_71001	-12001	The `dc_mcf_tdctcn()` function cannot be accepted because the MCF is under start processing.
DCMCFRTN_71002	-12002	The `dc_mcf_tdctcn()` function cannot be accepted because the MCF is under termination processing.

Return value	Return value (numeric)	Explanation
DCMCFRTN_71004	-12004	A memory shortage occurred during dc_mcf_tdctcn() function processing.
DCMCFRTN_71005	-12005	A communication error occurred. For the cause, see the message log file.
DCMCFRTN_71006	-12006	An internal error occurred. For the cause, see the message log file.
DCMCFRTN_71007	-12007	The specified connection name has not been registered.
DCMCFRTN_71008	-12008	The specified logical terminal name has not been registered.
DCMCFRTN_71009	-12009	The dc_mcf_tdctcn() function is not supported by the applicable MCF communication process.
DCMCFRTN_71010	-12010	Although the request to release the connection was issued to the MCF communication process, the request was not accepted. For the cause, see the message log file.
DCMCFRTN_71011	-12011	The dc_mcf_tdctcn() function cannot be accepted because the connection has been deleted.
DCMCFRTN_71014	-12014	The specified logical terminal name belongs to TP1/NET/NCSB or TP1/NET/X25-Extended; or, the specified connection group name belongs to TP1/NET/OSI-TP or TP1/NET/TCP/IP.
DCMCFRTN_72050	-13050	An unsupported flag is set in action.
DCMCFRTN_72051	-13051	NULL is set in cnopt.
DCMCFRTN_72052	-13052	When DCMCFPRO is not set in action: NULL is not set in proinf.
		When DCMCFPRO is set in action: A value smaller than 0 or a value 1025 or greater is specified for the size of protocol-specific area L pointed to by proinf.
DCMCFRTN_72053	-13053	NULL is not set in resv02.
DCMCFRTN_72054	-13054	NULL is not set in resv03.
DCMCFRTN_72055	-13055	NULL is not set in resv04.
DCMCFRTN_72060	-13060	DCMCFLE and DCMCFCN cannot be specified together in action.
DCMCFRTN_72061	-13061	A value smaller than 0 or a value 240 or greater is specified for mcfid in dcmcf_tdctcnopt.
DCMCFRTN_72062	-13062	resv01 in dcmcf_tdctcnopt is not filled with null characters.
DCMCFRTN_72063	-13063	idnam in dcmcf_tdctcnopt begins with the null character.

Return value	Return value (numeric)	Explanation
DCMCFRTN_72064	-13064	resv02 in dcmcf_tdctcnopt is not filled with null characters.
DCMCFRTN_72065	-13065	resv03 in dcmcf_tdctcnopt is not filled with null characters.
DCMCFRTN_72066	-13066	scnnam in dcmcf_tdctcnopt is not filled with null characters.
DCMCFRTN_72067	-13067	resv04 in dcmcf_tdctcnopt is not filled with null characters.
DCMCFRTN_72069	-13069	resv05 in dcmcf_tdctcnopt is not filled with null characters.
DCMCFRTN_72073	-13073	The character string set in idnam in dcmcf_tdctcnopt is 9 or more bytes in length.
DCMCFRTN_72074	-13074	The character string set in idnam in dcmcf_tdctcnopt contains an invalid character.

dc_mcf_tdctle - Shut down a logical terminal

Format

- ANSI C, C++

```
#include <dcmcf.h>
int dc_mcf_tdctle (DCLONG action, dcmcf_tdctleopt *leopt,
                   char *proinf, DCLONG *resv02,
                   char *resv03, char *resv04)
```

- K&R C

```
#include <dcmcf.h>
int dc_mcf_tdctle (action, leopt, proinf, resv02, resv03, resv04)
DCLONG             action;
dcmcf_tdctleopt    *leopt;
char               *proinf;
DCLONG             *resv02;
char               *resv03;
char               *resv04;
```

Description

The `dc_mcf_tdctle()` function shuts down a logical terminal.

Normal termination of the `dc_mcf_tdctle()` function indicates that the logical terminal shutdown request was accepted successfully by the protocol product. However, this does not necessarily mean that the logical terminal has been shut down.

If you intend to perform any operation related to the logical terminal after calling the `dc_mcf_tdctle()` function, first use the `dc_mcf_tlsle()` function to check the logical terminal's status.

Arguments whose values are set in the UAP

- `action`

Depending on the communication protocol, specify in one of the following formats the method used to specify for this function the logical terminal that is shut down:

`DCMCFLE[|DCMCFPRO]`

`DCMCFLE`

Specifies that the logical terminal name is used.

DCMCFPRO

Specifies that the function depends on the communication protocol being used.

- leopt

Set in a dcmcf_tdctleopt structure the information about the logical terminal that is to be the subject of this function's processing.

The following shows the format of the structure:

```
typedef struct {
                DCLONG    mcfid;          ...MCF communication
                                              process identifier
                char      resv01[4];      ...Reserved
                char      idnam[9];       ...Logical terminal name
                char      resv02[7];      ...Reserved
                char      resv03[112];    ...Reserved
                char      resv04[376];    ...Reserved
                } dcmcf_tdctleopt;
```

- mcfid

  Specify the MCF communication process identifier of the MCF communication service for the logical terminal that is to be processed. The permitted value range is from 0 to 239.

  If you specify 0, the system searches for the MCF communication service to which the specified logical terminal name belongs. In a configuration where many MCF communication services are running or when you issue this function many times from a UAP, we recommend that you specify the MCF communication process identifier.

- resv01

  Fill the area with null characters.

- idnam

  Specify the name of the logical terminal to be shut down. The logical terminal name must be specified as a maximum of 8 bytes of characters and must end with the null character.

- resv02, resv03, resv04

  Fill the areas with null characters.

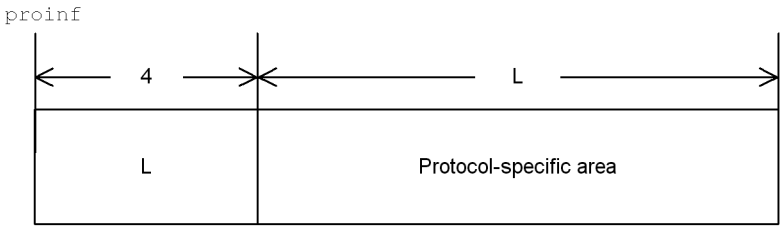
- proinf

Specify a protocol-specific area.

If you do not use a function that depends on the communication protocol, specify

NULL.

The following shows the format of a protocol-specific area:



The maximum size of a protocol-specific area is 1024 bytes.

The permitted value depends on the communication protocol being used. For details, see the applicable *OpenTP1 Protocol* manual.

- resv02, resv03, resv04

  Specify NULL.

Return values

Return value	Return value (numeric)	Explanation
DCMCFRTN_00000	0	Normal termination.
DCMCFRTN_71001	-12001	The dc_mcf_tdctle() function cannot be accepted because the MCF is under start processing.
DCMCFRTN_71002	-12002	The dc_mcf_tdctle() function cannot be accepted because the MCF is under termination processing.
DCMCFRTN_71004	-12004	A memory shortage occurred during dc_mcf_tdctle() function processing.
DCMCFRTN_71005	-12005	A communication error occurred. For the cause, see the message log file.
DCMCFRTN_71006	-12006	An internal error occurred. For the cause, see the message log file.
DCMCFRTN_71008	-12008	The specified logical terminal name has not been registered.
DCMCFRTN_71009	-12009	The dc_mcf_tdctle() function is not supported by the applicable MCF communication process.
DCMCFRTN_71010	-12010	Although the request to shut down the logical terminal was issued to the MCF communication process, the request was not accepted. For the cause, see the message log file.

Return value	Return value (numeric)	Explanation
DCMCFRTN_71011	-12011	The dc_mcf_tdctle() function cannot be accepted because the logical terminal has been deleted.
DCMCFRTN_72050	-13050	DCMCFLE is not set in action.
		An unsupported flag is set in action.
DCMCFRTN_72051	-13051	NULL is set in leopt.
DCMCFRTN_72052	-13052	When DCMCFPRO is not set in action: NULL is not set in proinf.
		When DCMCFPRO is set in action: A value smaller than 0 or a value 1025 or greater is specified for the size of protocol-specific area L pointed to by proinf.
DCMCFRTN_72053	-13053	NULL is not set in resv02.
DCMCFRTN_72054	-13054	NULL is not set in resv03.
DCMCFRTN_72055	-13055	NULL is not set in resv04.
DCMCFRTN_72061	-13061	A value smaller than 0 or a value 240 or greater is specified for mcfid in dcmcf_tdctleopt.
DCMCFRTN_72062	-13062	resv01 in dcmcf_tdctleopt is not filled with null characters.
DCMCFRTN_72063	-13063	idnam in dcmcf_tdctleopt begins with the null character.
DCMCFRTN_72064	-13064	resv02 in dcmcf_tdctleopt is not filled with null characters.
DCMCFRTN_72065	-13065	resv03 in dcmcf_tdctleopt is not filled with null characters.
DCMCFRTN_72067	-13067	resv04 in dcmcf_tdctleopt is not filled with null characters.
DCMCFRTN_72073	-13073	The character string set in idnam in dcmcf_tdctleopt is 9 or more bytes in length.
DCMCFRTN_72074	-13074	The character string set in idnam in dcmcf_tdctleopt contains an invalid character.

dc_mcf_tdlqle - Delete a logical terminal's output queue

Format

- ANSI C, C++

```
#include <dcmcf.h>
int dc_mcf_tdlqle (DCLONG action, dcmcf_tdlqleopt *leopt,
                   char *resv01, DCLONG *resv02,
                   char *resv03, char *resv04)
```

- K&R C

```
#include <dcmcf.h>
int dc_mcf_tdlqle (action, leopt, resv01, resv02, resv03, resv04)
DCLONG             action;
dcmcf_tdlqleopt    *leopt;
char               *resv01;
DCLONG             *resv02;
char               *resv03;
char               *resv04;
```

Description

The `dc_mcf_tdlqle()` function deletes a logical terminal's output queue.

When the function deletes the output queue successfully, it sends an event that reports that unprocessed send messages have been discarded (`ERREVTA`).

Arguments whose values are set in the UAP

- `action`

  Specify `DCMCFLE` to indicate that a logical terminal name is being specified.

- `leopt`

  Set in a `dcmcf_tdlqleopt` structure the connection information about the logical terminal that is to be the subject of this function's processing.

  The following shows the format of the structure:

```
typedef struct {
                 DCLONG     mcfid;          ...MCF communication
                                                process identifier
                 char       resv01[4];      ...Reserved
                 char       idnam[9];       ...Logical terminal name
                 char       resv02[7];      ...Reserved
                 char       resv03[112];    ...Reserved
```

```
char      resv04[376];   ...Reserved
} dcmcf_tdlqleopt;
```

- `mcfid`

  Specify the MCF communication process identifier of the MCF communication service for the logical terminal to be processed. The permitted value range is from `0` to `239`.

  If you specify `0`, the system searches for the MCF communication service to which the specified logical terminal name belongs. In a configuration where many MCF communication services are running or when you issue this function many times from a UAP, we recommend that you specify the MCF communication process identifier.

- `resv01`

  Fill the area with null characters.

- `idnam`

  Specify the name of the logical terminal whose output queue is deleted. The logical terminal name must be specified as a maximum of 8 bytes of characters and must end with the null character.

- `resv02`, `resv03`, `resv04`

  Fill the areas with null characters.

■ `resv01`, `resv02`, `resv03`, `resv04`

Specify `NULL`.

Return values

Return value	Return value (numeric)	Explanation
`DCMCFRTN_00000`	`0`	Normal termination.
`DCMCFRTN_71001`	`-12001`	The `dc_mcf_tdlqle()` function cannot be accepted because the MCF is under start processing.
`DCMCFRTN_71002`	`-12002`	The `dc_mcf_tdlqle()` function cannot be accepted because the MCF is under termination processing.
`DCMCFRTN_71004`	`-12004`	A memory shortage occurred during `dc_mcf_tdlqle()` function processing.
`DCMCFRTN_71005`	`-12005`	A communication error occurred. For the cause, see the message log file.

Return value	Return value (numeric)	Explanation
DCMCFRTN_71006	-12006	An internal error occurred. For the cause, see the message log file.
DCMCFRTN_71008	-12008	The specified logical terminal name has not been registered.
DCMCFRTN_71009	-12009	The dc_mcf_tdlqle() function is not supported by the applicable MCF communication process.
DCMCFRTN_71010	-12010	Although the request to delete the logical terminal's output queue was issued to the MCF communication process, the request was not accepted. For the cause, see the message log file.
DCMCFRTN_71011	-12011	The dc_mcf_tdlqle() function cannot be accepted because the logical terminal has been deleted.
DCMCFRTN_71017	-12017	The dc_mcf_tdlqle() function cannot be accepted because the logical terminal has not been shut down.
DCMCFRTN_71018	-12018	The dc_mcf_tdlqle() function cannot be accepted because the session has not been closed.
DCMCFRTN_71019	-12019	The dc_mcf_tdlqle() function cannot be accepted because an alternate send operation is underway.
DCMCFRTN_72050	-13050	DCMCFLE is not set in action.
		An unsupported flag is set in action.
DCMCFRTN_72051	-13051	NULL is set in leopt.
DCMCFRTN_72052	-13052	NULL is not set in resv01.
DCMCFRTN_72053	-13053	NULL is not set in resv02.
DCMCFRTN_72054	-13054	NULL is not set in resv03.
DCMCFRTN_72055	-13055	NULL is not set in resv04.
DCMCFRTN_72061	-13061	A value smaller than 0 or a value 240 or greater is specified for mcfid in dcmcf_tdlqleopt.
DCMCFRTN_72062	-13062	resv01 in dcmcf_tdlqleopt is not filled with null characters.
DCMCFRTN_72063	-13063	idnam in dcmcf_tdlqleopt begins with the null character.
DCMCFRTN_72064	-13064	resv2 in dcmcf_tdlqleopt is not filled with null characters.
DCMCFRTN_72065	-13065	resv03 in dcmcf_tdlqleopt is not filled with null characters.
DCMCFRTN_72067	-13067	resv04 in dcmcf_tdlqleopt is not filled with null characters.

Return value	Return value (numeric)	Explanation
DCMCFRTN_72073	-13073	The character string set in idnam in dcmcf_tdlqleopt is 9 or more bytes in length.
DCMCFRTN_72074	-13074	The character string set in idnam in dcmcf_tdlqleopt contains an invalid character.

dc_mcf_tempget - Accept temporary-stored data

Format

- ANSI C, C++

```
#include <dcmcf.h>}
int  dc_mcf_tempget(DCLONG action,char *getdata,DCLONG gtempleng,
                    DCLONG *gdataleng,char *resv01)
```

- K&R C

```
#include <dcmcf.h>
int  dc_mcf_tempget (action,getdata,gtempleng,gdataleng,resv01)
DCLONG       action;
char        *getdata;
DCLONG       gtempleng;
DCLONG       *gdataleng;
char        *resv01;
```

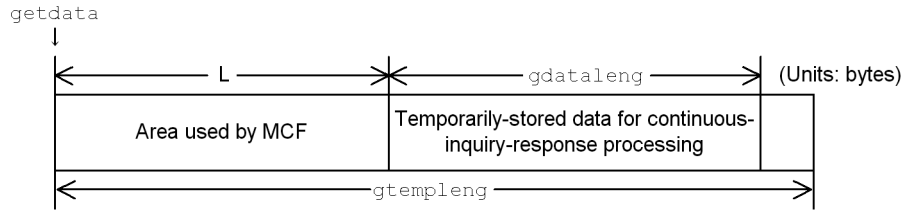
Description

The function `dc_mcf_tempget()` receives data stored in the temporary-stored area which is used for continuous-inquiry-response processing.

For `gtempleng`, specify a value from 1 to 32,000 bytes. If the temporary-stored data exceeds the length specified for `gtempleng`, the excess portion is truncated. If the temporary-stored data is shorter than `gtempleng -8` (with buffer format 1) or `gtempleng -6` (with buffer format 2), no processing is done for the remaining receive area.

If there is no temporary-stored data, the function `dc_mcf_tempget()` is executed on the assumption that $(00)_{16}$ equivalent to the length specified for tempsize in the MCF application definition is specified.

The figure below shows the format of the receive segment area. With buffer format 1, L is 8 bytes; with buffer format 2, L is 6 bytes.



Arguments whose values are set in the UAP

- `action`

  Specify the type of buffer format to be used.

```
{DCMCFBUF1|DCMCFBUF2}
```

  `DCMCFBUF1`

  Specify `DCMCFBUF1` when using buffer format 1.

  `DCMCFBUF2`

  Specify `DCMCFBUF2` when using buffer format 2.

- `getdata`

  Specify the area for receiving temporary-stored data. After the function `dc_mcf_tempget()` is called, the temporary-stored data is returned to the area indicated by getdata.

- `gtempleng`

  Specify the length of the area for receiving temporary-stored data. The number of bytes to be specified varies depending on the buffer format.

- `resv01`

  Specify a null character.

Arguments whose values are returned from OpenTP1

- `getdata`

  The temporary-stored data is returned.

- `gdataleng`

  The length of previously updated data is returned.

Return values

Return value	Return value (numeric)	Explanation
DCMCFRTN_00000	0	Normal termination.
DCMCFRTN_72000	-13000	The function `dc_mcf_tempget()` cannot be called from an SPP.
DCMCFRTN_72013	-13013	Temporary-stored data exceeding the length of the receive area was received. The excess portion was truncated.

Return value	Return value (numeric)	Explanation
DCMCFRTN_72016	-13016	The value specified for action is invalid.
		The value of the area pointed to by resv01 is not a null character.
DCMCFRTN_72036	-13036	The receive area length is less than 9 bytes (with buffer format 1) or less than 7 bytes (with buffer format 2).
DCMCFRTN_72101	-13101	The function dc_mcf_tempget() was called from an MHP for which type=cont (continuous-inquiry-response type) was not specified in the MCF application definition.
DCMCFRTN_72106	-13106	The function dc_mcf_tempget() was called before the function dc_mcf_receive() for receiving the first segment.
DCMCFRTN_72107	-13107	The function dc_mcf_tempget() was called after the function dc_mcf_contend().
Other than the above		An unprecedented error (e.g., program damage) occurred.

dc_mcf_tempput - Update temporary-stored data

Format

- ANSI C, C++

```
#include <dcmcf.h>
int  dc_mcf_tempput(DCLONG action,char *putdata,DCLONG pdataleng,
                    char *resv01)
```

- K&R C

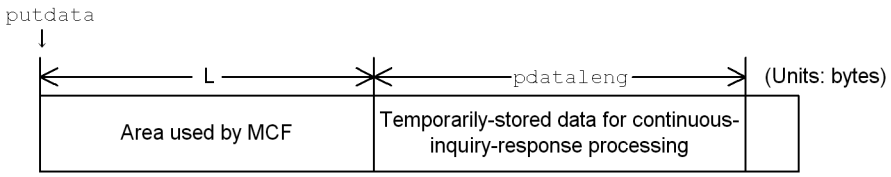
```
#include <dcmcf.h>
int  dc_mcf_tempput (action, putdata, pdataleng, resv01)
DCLONG    action;
char      *putdata;
DCLONG    pdataleng;
char      *resv01;
```

Description

The function `dc_mcf_tempput()` updates data stored in the temporary-stored area which is used for continuous-inquiry-response processing.

Call the function `dc_mcf_tempget()` before the function `dc_mcf_tempput()`.

The figure below shows the format of the send segment area. With buffer format 1, L is 8 bytes; with buffer format 2, L is 6 bytes.



Arguments whose values are set in the UAP

- `action`

Specify the type of buffer format to be used.

```
{DCMCFBUF1|DCMCFBUF2}
```

`DCMCFBUF1`

Specify that buffer format 1 is used.

`DCMCFBUF2`

Specify that buffer format 2 is used.

- `putdata`

  Specify the area storing the temporary-stored data to be updated.

- `ptempleng`

  Specify the length of the temporary-stored data to be updated.

- `resv01`

  Specify a null character.

Return values

Return value	Return value (numeric)	Explanation
DCMCFRTN_00000	0	Normal termination.
DCMCFRTN_71103	-12103	The area for updating the temporary-stored data could not be acquired.
DCMCFRTN_72000	-13000	The function `dc_mcf_tempput()` cannot be called from an SPP.
DCMCFRTN_72016	-13016	The value specified for `action` is invalid.
		The value of the area pointed to by `resv01` is not a null character.
DCMCFRTN_72035	-13035	The value specified for the data update length exceeds the value specified for the temporary data storage area length in the MCF application definition.
		The value specified for the data update length is less than 1 byte.
DCMCFRTN_72101	-13101	The function `dc_mcf_tempput()` was called from an MHP for which `type=cont` (continuous-inquiry-response type) was not specified in the MCF application definition.
DCMCFRTN_72105	-13105	The function `dc_mcf_tempput()` was called before the function `dc_mcf_tempget()`.
DCMCFRTN_72106	-13106	The function `dc_mcf_tempput()` was called before the function `dc_mcf_receive()` for receiving the first segment.
DCMCFRTN_72107	-13107	The function `dc_mcf_tempput()` was called after the function `dc_mcf_contend()`.

Return value	Return value (numeric)	Explanation
Other than the above		An unprecedented error (e.g., program damage) occurred.

dc_mcf_timer_cancel - Cancel user timer monitoring

Format

- ANSI C, C++

```
#include <dcmcf.h>
int dc_mcf_timer_cancel(DCLONG flags,DCLONG timer_id,char *lename)
```

- K&R C

```
#include <dcmcf.h>
int dc_mcf_timer_cancel(flags,timer_id,lename)
DCLONG    flags;
DCLONG    timer_id;
char      *lename;
```

Description

The function `dc_mcf_timer_cancel()` cancels user timer monitoring that was set by the function `dc_mcf_timer_set()`.

This function cancels user timer monitoring as soon as the function `dc_mcf_timer_cancel()` returns normally.

If user timer monitoring has reached timeout and an MHP has already been started at the time this function is called, the function `dc_mcf_timer_cancel()` returns with the error `DCMCFER_PARAM_TIM_ID`.

Only a user server can call the function `dc_mcf_timer_cancel()`.

Arguments whose values are set in the UAP

- `flags`

  Specify `DCNOFLAGS`.

- `timer_id`

  Specify the same timer request identifier as that specified when user timer monitoring was set.

- `lename`

  Specify the same logical terminal name as that specified when user timer monitoring was set.

Return values

Return value	Return value (numeric)	Explanation
DC_OK	0	Normal termination.
DCMCFER_PARAM_FLAGS	-11911	The value specified for `flags` is invalid.
DCMCFER_PARAM_TIM_ID	-11910	The timer request identifier specified for `timer_id` is not registered.
		Timeout already occurred and the MHP has started, or user timer monitoring was already canceled.
DCMCFER_PARAM_LENAME	-11912	The value specified for `lename` is invalid.
DCMCFER_NO_DEFINE	-11916	The requested facility is not defined in the MCF.

dc_mcf_timer_set - Set user timer monitoring

Format

- ANSI C, C++

```
#include <dcmcf.h>
int dc_mcf_timer_set(DCLONG flags,DCLONG timer,DCLONG timer_id,
                     char *lename,char *apname,char *data,DCLONG data_leng,
                     DCLONG resv01,DCLONG resv02)
```

- K&R C

```
#include <dcmcf.h>
int dc_mcf_timer_set (flags, timer, timer_id, lename,
                      apname, data, data_leng, resv01,
                      resv02)
DCLONG    flags;
DCLONG    timer;
DCLONG    timer_id;
char      *lename;
char      *apname;
char      *data;
DCLONG    data_leng;
DCLONG    resv01;
DCLONG    resv02;
```

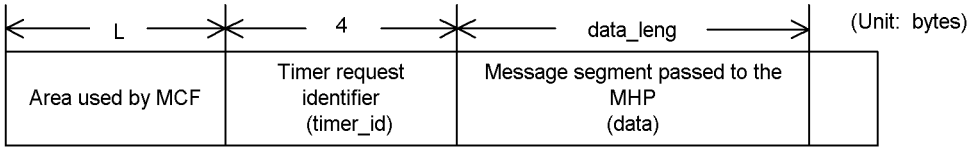
Description

Use the function `dc_mcf_timer_set()` from a UAP to set user timer monitoring for monitoring the desired interval. To call this function, you must specify `usertime=yes` in the `-p` option of the MCF communication configuration definition `mcfttim`.

Only a user server can call the function `dc_mcf_timer_set()`.

When the time (in seconds) specified for `timer` elapses (when timeout occurs), the logical terminal specified for `lename` generates an event and starts the MHP having the application name specified for `apname`. You can omit `lename` only when the UAP is an MHP. In this case, the input source logical terminal is assumed.

The MHP to be started when timeout occurs must be a non-response-type (`noans` type) MHP. The figure below shows the format of the message segment when a message is passed to the MHP. With buffer format 1, L is 8 bytes; with buffer format 2, L is 4 bytes.



To cancel the user timer monitor set by the function `dc_mcf_timer_set()`, call the function `dc_mcf_timer_cancel()` with the same values specified for `timer_id` and `lename` as specified in the function `dc_mcf_timer_set()`.

The time monitor starts as soon as the function `dc_mcf_timer_set()` is called.

The maximum number of time monitors you can run concurrently is indicated by the maximum number of time monitoring requests specified for the timereqno operand in the `-p` option of the MCF communication configuration definition `mcfttim`.

Arguments whose values are set in the UAP

- `flags`

  Specify `DCNOFLAGS`.

- `timer`

  Specify the number of seconds that are to elapse before the MHP is started after the function `dc_mcf_timer_set()` is called. The specifiable range is 1 to 360000 (from 1 second to 100 hours).

  Since OpenTP1 monitors timeout at fixed intervals, an error arises between the time specified for `timer` and the time that elapses before actual detection of timeout. The accuracy of time monitoring depends on the value of the interval of time monitoring specified for the `btim` operand in the `-t` option of the MCF communication configuration definition `mcfttim`.

- `timer_id`

  Specify the timer request identifier.

  `timer_id` provides information for identifying this timer. Be sure to specify a value for `timer_id` that is unique in the logical terminal specified for `lename`.

- `lename`

  Specify in 8 or fewer bytes the name of the logical terminal that is to generate an event when timeout occurs. Append a null character to the end of the logical terminal name. When omitting this value, specify a null character. The default is the input source logical terminal.

- `apname`

  Specify the application name of the MHP to be started. The attribute of this application

must be defined in the application attribute definition (mcfaalcap) field within the MCF application definition that is specified by the -a option to the MCF communication configuration definition mcftenv. This MCF communication configuration definition (mcftenv) is for the MCF communication server that serves the logical terminal specified by lename. server having the logical terminal name specified for lename. Specify the application name in up to 8 bytes. Append a null character to the end of the application name. The MHP must be a non-response-type (noans type) MHP. The specified application name must be a user event.

- data

  Specify the contents of the message segment to be passed to the MHP to be started. You cannot specify multiple segments. If no segment is to be passed to the MHP to be started, specify a null character.

- data_leng

  Specify the length of the segment to be passed to the MHP to be started. If no segment is to be passed to the MHP to be started, specify 0.

  The specifiable range is 0 to 256. The maximum specifiable value depends on the maximum message length specified for the msgsize operand in the

  -p option of the MCF communication configuration definition mcfttim.

- resv01

  Specify DCNOFLAGS.

- resv02

  Specify DCNOFLAGS.

Return values

Return value	Return value (numeric)	Explanation
DC_OK	0	Normal termination.
DCMCFER_PARAM_FLAGS	-11911	The value specified for flags is invalid.
DCMCFER_PARAM_LENAME	-11912	The value specified for lename is invalid.
DCMCFER_PARAM_TIMER	-11909	The value specified for timer is invalid.
DCMCFER_PARAM_APNAME	-11913	The value specified for apname is invalid.
DCMCFER_PARAM_DATA	-11915	The value specified for data is invalid.
DCMCFER_PARAM_LENG	-11914	The value specified for data_leng is invalid.

Return value	Return value (numeric)	Explanation
DCMCFER_PARAM_TIM_ID	-11910	The timer request identifier specified for timer_id is already registered.
DCMCFER_INVALID_ARGS	-11900	The value specified for an argument is invalid.
DCMCFER_NO_DEFINE	-11916	The requested facility is not defined in the MCF.
DCMCFER_NO_TIMER_ENT	-11917	User timer monitoring cannot be set because there is no free space in the timer registration area. To reserve the timer registration area, revise the value of the timereqno operand in the -p option of the MCF communication configuration definition mcfttim. If required, check the values of the -p option of the MCF manager definition mcfmcomn and the static_shmpool_size operand in the system environment definition.

dc_mcf_tlscn - Acquire a connection status

Format

- ANSI C, C++

```
#include <dcmcf.h>
int dc_mcf_tlscn (DCLONG action, dcmcf_tlscnopt *cnopt,
                  char *resv01, DCLONG *resv02,
                  char *resv03, DCLONG *infcnt,
                  dcmcf_cninf *inf, char *resv04)
```

- K&R C

```
#include <dcmcf.h>
int dc_mcf_tlscn (action, cnopt, resv01, resv02, resv03, infcnt,
                  inf, resv04)
DCLONG            action;
dcmcf_tlscnopt    *cnopt;
char              *resv01;
DCLONG            *resv02;
char              *resv03;
DCLONG            *infcnt;
dcmcf_cninf       *inf;
char              *resv04;
```

Description

The `dc_mcf_tlscn()` function acquires the status of a connection.

Arguments whose values are set in the UAP

- `action`

Specify in one of the following formats the method used to specify the connection whose status is to be acquired:

{DCMCFLE|DCMCFCN}

`DCMCFLE`

Specifies that a logical terminal name is specified for the connection whose status is to be acquired.

This argument is not supported by TP1/NET/NCSB or TP1/NET/X25-Extended.

`DCMCFCN`

Specifies that a connection ID is specified for the connection whose status is to be acquired.

- `cnopt`

  Set in a `dcmcf_tlscnopt` structure the information about the connection that is to be the subject to this function's processing.

  The following shows the format of the structure:

```
typedef struct {
                DCLONG    mcfid;         ...MCF communication
                                                process identifier
                char      resv01[4];     ...Reserved
                char      idnam[9];      ...Logical terminal name
                                                or connection ID
                char      resv02[7];     ...Reserved
                char      resv03[112];   ...Reserved
                char      resv04[376];   ...Reserved
                } dcmcf_tlscnopt;
```

  - `mcfid`

    Specify the MCF communication process identifier of the MCF communication service for the connection to be processed. The permitted value range is from `0` to `239`.

    This argument is ignored when a logical terminal name is used to request connection status acquisition.

    If you specify `0`, the system searches for the MCF communication service to which the specified connection ID belongs. In a configuration where many MCF communication services are running or when you issue this function many times from a UAP, we recommend that you specify the MCF communication process identifier.

  - `resv01`

    Fill the area with null characters.

  - `idnam`

    Specify the logical terminal name or connection ID of the connection whose status is to be acquired. The logical terminal name or connection ID must be specified as a maximum of 8 bytes of characters and must end with the null character.

  - `resv02`, `resv03`, `resv04`

    Fill the areas with the null characters.

- resv01, resv02, resv03

  Specify NULL.

- infcnt

  Specify 1 as the number of dcmcf_cninf areas to be used to store connection status.

  When the processing is completed, the number of corresponding connections is returned.

- inf

  Specify the dcmcf_cninf area for storing the connection status.

  The size of this area must be at least the size of the dcmcf_cninf structure x infcnt.

- resv04

  Specify NULL.

Arguments whose values are returned from OpenTP1

- infcnt

  Returns the number of connections that were processed by this function.

- inf

  Returns the dcmcf_cninf structure containing the information about the connection that was processed by this function.

  The following shows the format of the structure:

```
typedef struct {
                char     idnam[9];       ...Connection ID
                char     resv01[7];      ...Reserved
                char     pnam[4];        ...Protocol type
                DCLONG   status;         ...Connection status
                char     resv02[40];     ...Reserved
                } dcmcf_cninf;
```

  - idnam

    Sets the connection ID of the requested connection.

  - resv01

    Fills the area with null characters.

  - pnam

    Sets one of the following values as the protocol type of the requested connection:

'UA '
: TP1/NET/User Agent (OSAS/UA protocol)

'hds'
: TP1/NET/HDLC (HDLC protocol)

'X25'
: TP1/NET/X25 (X.25 protocol)

'TP '
: TP1/NET/OSI-TP (OSI TP protocol)

'XP '
: TP1/NET/XMAP3

'HS1'
: TP1/NET/HSC (HSC1 protocol)

'HS2'
: TP1/NET/HSC (HSC2 protocol)

'CSB'
: TP1/NET/NCSB (NCSB protocol)

'NIF'
: TP1/NET/OSAS-NIF (NIF protocol)

'SL2'
: TP1/NET/Secondary Logical Unit - TypeP2 (SLUTYPE-P protocol (secondary station))

'TCP'
: TP1/NET/TCP/IP (TCP/IP protocol)

'X2E'
: TP1/NET/X25-Extended (X.25 protocol)

- status

  Sets one of the following values as the status of the requested connection:

  DCMCF_CNST_ACT
  : A connection has been established.

  DCMCF_CNST_ACT_B

Connection establishment processing is underway.

DCMCF_CNST_DCT

A connection has been released.

DCMCF_CNST_DCT_B

Connection release processing is underway.

- resv02

  Fills the area with null characters.

Return values

Return value	Return value (numeric)	Explanation
DCMCFRTN_00000	0	Normal termination.
DCMCFRTN_71001	-12001	The dc_mcf_tlscn() function cannot be accepted because the MCF is under start processing.
DCMCFRTN_71004	-12004	A memory shortage occurred during dc_mcf_tlscn() function processing.
DCMCFRTN_71005	-12005	A communication error occurred. For the cause, see the message log file.
DCMCFRTN_71006	-12006	An internal error occurred. For the cause, see the message log file.
DCMCFRTN_71007	-12007	The specified connection name has not been registered.
DCMCFRTN_71008	-12008	The specified logical terminal name has not been registered.
DCMCFRTN_71009	-12009	The dc_mcf_tlscn() function is not supported by the applicable MCF communication process.
DCMCFRTN_71010	-12010	Although the request to acquire the connection status was issued to the MCF communication process, the request was not accepted. For the cause, see the message log file.
DCMCFRTN_71011	-12011	The dc_mcf_tlscn() function cannot be accepted because the connection has been deleted.
DCMCFRTN_71014	-12014	The specified logical terminal name belongs to TP1/NET/NCSB or TP1/NET/X25-Extended; or, the specified connection group name belongs to TP1/NET/OSI-TP or TP1/NET/TCP/IP.
DCMCFRTN_72050	-13050	An unsupported flag is set in action.
DCMCFRTN_72051	-13051	NULL is set in cnopt.
DCMCFRTN_72052	-13052	NULL is not set in resv01.

Return value	Return value (numeric)	Explanation
DCMCFRTN_72053	-13053	NULL is not set in resv02.
DCMCFRTN_72054	-13054	NULL is not set in resv03.
DCMCFRTN_72055	-13055	NULL is not set in resv04.
DCMCFRTN_72056	-13056	NULL is set in infcnt.
DCMCFRTN_72057	-13057	NULL is set in inf.
DCMCFRTN_72060	-13060	DCMCFLE and DCMCFCN cannot be specified together in action.
DCMCFRTN_72061	-13061	A value smaller than 0 or a value 240 or greater is specified for mcfid in dcmcf_tlscnopt.
DCMCFRTN_72062	-13062	resv01 in dcmcf_tlscnopt is not filled with null characters.
DCMCFRTN_72063	-13063	idnam in dcmcf_tlscnopt begins with the null character.
DCMCFRTN_72064	-13064	resv02 in dcmcf_tlscnopt is not filled with null characters.
DCMCFRTN_72065	-13065	resv03 in dcmcf_tlscnopt is not filled with null characters.
DCMCFRTN_72067	-13067	resv04 in dcmcf_tlscnopt is not filled with null characters.
DCMCFRTN_72073	-13073	The character string set in idnam in dcmcf_tlscnopt is 9 or more bytes in length.
DCMCFRTN_72074	-13074	The character string set in idnam in dcmcf_tlscnopt contains an invalid character.
DCMCFRTN_72076	-13076	The value 1 is not set in infcnt.

dc_mcf_tlscom - Acquire the status of MCF communication services

Format

- ANSI C, C++

```
#include <dcmcf.h>
int dc_mcf_tlscom (DCLONG action, char *resv01, DCLONG *infcnt,
                   dcmcf_svinf *inf, char *resv02)
```

- K&R C

```
#include <dcmcf.h>
int dc_mcf_tlscom (action, resv01, infcnt, inf, resv02)
DCLONG             action;
char               *resv01;
DCLONG             *infcnt;
dcmcf_svinf        *inf;
char               *resv02;
```

Description

The `dc_mcf_tlscom()` function acquires the statuses of the MCF communication services or application start services.

Arguments whose values are set in the UAP

- `action`

  Specify `DCNOFLAGS`.

- `resv01`

  Specify `NULL`.

- `infcnt`

  Specify the number of `dcmcf_svinf` areas used to store the statuses of the MCF communication services or application start services.

  When the processing is completed, the number of corresponding MCF communication services is returned.

- `inf`

  Specify the `dcmcf_svinf` area used to store the statuses of the MCF communication services or application start services.

  The size of this area must be at least the size of the `dcmcf_svinf` structure x `infcnt`.

- resv02

  Specify NULL.

Arguments whose values are returned from OpenTP1

- infcnt

  Returns the number of application start services or MCF communication services that have been registered in the MCF service.

- inf

  Returns the dcmcf_svinf structure that contains information about the application start services or MCF communication services registered in the MCF service.

  The following shows the format of the structure:

```
typedef struct {
                DCLONG     mcfid;        ...MCF communication
                                              process identifier or
                                              Application start
                                              process identifier
                char       svname[9];    ...MCF communication
                                              service name
                char       resv01[7];    ...Reserved
                char       pnam[20];     ...Protocol type
                DCLONG     status;       ...Status of MCF
                                              communication service
                char       resv02[20];   ...Reserved
                } dcmcf_svinf;
```

  - mcfid

    Sets an application start process identifier or MCF communication process identifier.

  - svname

    Sets the MCF communication service name.

  - resv01

    Fills the area with null characters.

  - pnam

    Sets the protocol type.

    'MCF ΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔ'

    Application start service for TP1/Message Control

'User Δ Agent ΔΔΔΔΔΔΔΔΔΔ '

TP1/NET/User Agent (OSAS/UA protocol)

'HDLC ΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔ '

TP1/NET/HDLC (HDLC protocol)

'X25 ΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔ '

TP1/NET/X25 (X.25 protocol)

'TP ΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔ '

TP1/NET/OSI-TP (OSI TP protocol)

'XMAP3 ΔΔΔΔΔΔΔΔΔΔΔΔΔΔ '

TP1/NET/XMAP3

'HSC ΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔ '

TP1/NET/HSC (HSC protocol)

'NCSB ΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔ '

TP1/NET/NCSB (NCSB protocol)

'OSAS-NIF ΔΔΔΔΔΔΔΔΔΔΔΔ '

TP1/NET/OSAS-NIF (NIF/OSI protocol)

'NET/SLUP2 ΔΔΔΔΔΔΔΔΔΔ '

TP1/NET/Secondary Logical Unit - TypeP2 (SLUTYPE-P protocol (secondary station))

'TCP/IP ΔΔΔΔΔΔΔΔΔΔΔΔΔ '

TP1/NET/TCP/IP (TCP/IP protocol)

'X25-EX ΔΔΔΔΔΔΔΔΔΔΔΔΔ '

TP1/NET/X25-Extended (X.25 protocol)

'UDP/IP ΔΔΔΔΔΔΔΔΔΔΔΔΔ '

TP1/NET/User Datagram Protocol (UDP protocol)

- status

  Sets one of the following values as the status of the MCF communication service or application start service:

  DCMCF_SVST_OFLN

  Service is stopped.

DCMCF_SVST_START

Service is under preparation processing.

DCMCF_SVST_ONLN

Service has started or is under preparation processing for termination.

DCMCF_SVST_PREEND

Service is under preparation processing for terminating partial stop.

DCMCF_SVST_END

Service is under stop processing.

- resv02

  Fills the area with null characters.

Return values

Return value	Return value (numeric)	Explanation
DCMCFRTN_00000	0	Normal termination.
DCMCFRTN_71001	-12001	The dc_mcf_tlscom() function cannot be accepted because the MCF is under start processing.
DCMCFRTN_71004	-12004	A memory shortage occurred during dc_mcf_tlscom() function processing.
DCMCFRTN_71005	-12005	A communication error occurred. For the cause, see the message log file.
DCMCFRTN_71006	-12006	An internal error occurred. For the cause, see the message log file.
DCMCFRTN_72013	-13013	The number of MCF communication services or application start services exceeded the value specified in infcnt. Information about the excess services was discarded.
DCMCFRTN_72050	-13050	DCNOFLAGS is not set in action.
DCMCFRTN_72052	-13052	NULL is not set in resv01.
DCMCFRTN_72053	-13053	NULL is not set in resv02.
DCMCFRTN_72056	-13056	NULL is set in infcnt.
DCMCFRTN_72057	-13057	NULL is set in inf.
DCMCFRTN_72076	-13076	A value of 0 or smaller is set in infcnt.

dc_mcf_tlsle - Acquire a logical terminal status

Format

- ANSI C, C++

```
#include <dcmcf.h>
int dc_mcf_tlsle (DCLONG action, dcmcf_tlsleopt *leopt,
                  char *resv01, DCLONG *resv02,
                  char *resv03, DCLONG *infcnt,
                  dcmcf_leinf2 *inf, char *resv04)
```

- K&R C

```
#include <dcmcf.h>
int dc_mcf_tlsle (action, leopt, resv01, resv02, resv03, infcnt,
                  inf, resv04)
DCLONG            action;
dcmcf_tlsleopt    *leopt;
char              *resv01;
DCLONG            *resv02;
char              *resv03;
DCLONG            *infcnt;
dcmcf_leinf2      *inf;
char              *resv04;
```

Description

The `dc_mcf_tlsle()` function acquires the status of a logical terminal.

Arguments whose values are set in the UAP

- `action`

  Specify `DCMCFLE` to indicate that a logical terminal name is to be specified.

- `leopt`

  Set in a `dcmcf_tlsleopt` structure the connection information about the logical terminal that is to be the subject to this function's processing.

  The following shows the format of the structure:

```
typedef struct {
                  DCLONG    mcfid;          ...MCF communication
                                                process identifier
                  char      resv01[4];      ...Reserved
                  char      idnam[9];       ...Logical terminal name
```

```
char       resv02[7];     ...Reserved
char       resv03[112];   ...Reserved
char       resv04[376];   ...Reserved
} dcmcf_tlsleopt;
```

- `mcfid`

  Specify the MCF communication process identifier of the MCF communication service for the logical terminal that is to be processed. The permitted value range is from `0` to `239`.

  If you specify `0`, the system searches for the MCF communication service to which the specified logical terminal name belongs. In a configuration where many MCF communication services are running or when you issue this function many times from a UAP, we recommend that you specify the MCF communication process identifier.

- `resv01`

  Fill the area with null characters.

- `idnam`

  Specify the name of the logical terminal whose status is to be acquired. The logical terminal name must be specified as a maximum of 8 bytes of characters and must end with the null character.

- `resv02`, `resv03`, `resv04`

  Fill the areas with null characters.

- `resv01`, `resv02`, `resv03`

  Specify `NULL`.

- `infcnt`

  Specify `1` as the number of `dcmcf_leinf2` areas for storing the logical terminal status.

  When the processing is completed, the number of corresponding logical terminals is returned.

- `inf`

  Specify a `dcmcf_leinf2` area for storing the logical terminal status information.

  The size of this area must be at least the size of the `dcmcf_leinf2` structure x `infcnt`.

- `resv04`

  Specify `NULL`.

Arguments whose values are returned from OpenTP1

- infcnt

  Returns the number of logical terminals that were processed by this function.

- inf

  Returns the dcmcf_leinf2 structure containing the information about the logical terminal that was processed by this function.

  The following shows the format of the structure:

```
typedef struct {
                char      idnam[9];     ...Logical terminal name
                char      resv01[7];    ...Reserved
                char      resv02[4];    ...Reserved
                DCLONG    status;       ...Logical terminal status
                char      resv03[40];   ...Reserved
                } dcmcf_leinf2;
```

  - idnam

    Sets the name of the requested logical terminal.

  - resv01, resv02

    Fills the areas with null characters.

  - status

    Sets one of the following values as the status of the requested logical terminal:

    DCMCF_LEST_ACT

    Logical terminal has been released from shutdown status.

    DCMCF_LEST_DCT

    Logical terminal has been shut down.

  - resv03

    Fills the area with null characters.

Return values

Return value	Return value (numeric)	Explanation
DCMCFRTN_00000	0	Normal termination.
DCMCFRTN_71001	-12001	The dc_mcf_tlsle() function cannot be accepted because the MCF is under start processing.

Return value	Return value (numeric)	Explanation
DCMCFRTN_71004	-12004	A memory shortage occurred during dc_mcf_tlsle() function processing.
DCMCFRTN_71005	-12005	A communication error occurred. For the cause, see the message log file.
DCMCFRTN_71006	-12006	An internal error occurred. For the cause, see the message log file.
DCMCFRTN_71008	-12008	The specified logical terminal name has not been registered.
DCMCFRTN_71009	-12009	The dc_mcf_tlsle() function is not supported by the applicable MCF communication process.
DCMCFRTN_71010	-12010	Although the request to acquire the logical terminal status was issued to the MCF communication process, the request was not accepted. For the cause, see the message log file.
DCMCFRTN_71011	-12011	The dc_mcf_tlsle() function cannot be accepted because the logical terminal has been deleted.
DCMCFRTN_72050	-13050	DCMCFLE is not set in action.
		An unsupported flag is set in action.
DCMCFRTN_72051	-13051	NULL is set in leopt.
DCMCFRTN_72052	-13052	NULL is not set in resv01.
DCMCFRTN_72053	-13053	NULL is not set in resv02.
DCMCFRTN_72054	-13054	NULL is not set in resv03.
DCMCFRTN_72055	-13055	NULL is not set in resv04.
DCMCFRTN_72056	-13056	NULL is set in infcnt.
DCMCFRTN_72057	-13057	NULL is set in inf.
DCMCFRTN_72061	-13061	A value smaller than 0 or a value 240 or greater is specified for mcfid in dcmcf_tlsleopt.
DCMCFRTN_72062	-13062	resv01 in dcmcf_tlsleopt is not filled with null characters.
DCMCFRTN_72063	-13063	idnam in dcmcf_tlsleopt begins with the null character.
DCMCFRTN_72064	-13064	resv02 in dcmcf_tlsleopt is not filled with null characters.
DCMCFRTN_72065	-13065	resv03 in dcmcf_tlsleopt is not filled with null characters.
DCMCFRTN_72067	-13067	resv04 in dcmcf_tlsleopt is not filled with null characters.

Return value	Return value (numeric)	Explanation
DCMCFRTN_72073	-13073	The character string set in idnam in dcmcf_tlsleopt is 9 bytes or more in length.
DCMCFRTN_72074	-13074	The character string set in idnam in dcmcf_tlsleopt contains an invalid character.
DCMCFRTN_72076	-13076	The value 1 is not set in infcnt.

dc_mcf_tlsln - Acquire the acceptance status for a server-type connection establishment request

Format

- ANSI C, C++

```
#include <dcmcf.h>
int dc_mcf_tlsln (DCLONG action, DCLONG mcfid, char *resv01,
                  DCLONG *infcnt, dcmcf_lninf *inf, char *resv02)
```

- K&R C

```
#include <dcmcf.h>
int dc_mcf_tlsln (action, mcfid, resv01, infcnt, inf, resv02)
DCLONG         action;
DCLONG         mcfid;
char           *resv01;
DCLONG         *infcnt;
dcmcf_lninf    *inf;
char           *resv02;
```

Description

The `dc_mcf_tlsln()` function acquires the acceptance status for a server-type connection establishment request.

Arguments whose values are set in the UAP

- `action`

  Set `DCNOFLAGS`.

- `mcfid`

  Specify the MCF communication process identifier of the MCF communication service that is to be processed. The permitted value range is from `1` to `239`.

- `resv01`

  Specify `NULL`.

- `infcnt`

  Specify `1` as the number of `dcmcf_lninf` areas to be used to store the acceptance status of the server-type connection establishment request.

  When the processing is completed, the number of corresponding MCF communication services is returned.

- inf

  Specify a dcmcf_lninf area to be used to store the acceptance status of the server-type connection establishment request.

  The size of this area must be at least the size of the dcmcf_lninf structure x infcnt.

- resv02

  Specify NULL.

Arguments whose values are returned from OpenTP1

- infcnt

  Returns the number of MCF communication services that were processed by this function.

- inf

  Returns the dcmcf_lninf structure containing the acceptance status of the server-type connection establishment request for the MCF communication service that was processed by this function.

  The following shows the format of the structure:

```
typedef struct {
                DCLONG    status;       ...Acceptance status
                char      resv01[60];   ...Reserved
                } dcmcf_lninf;
```

  - status

    Sets one of the following values as the acceptance status of the server-type connection establishment request:

    DCMCF_LNST_LISTEN

    The acceptance process for the server-type connection establishment request has started.

    DCMCF_LNST_RETRY

    The acceptance process for the server-type connection establishment request is under start processing.

    DCMCF_LNST_ONLN_W

    The acceptance process for the server-type connection establishment request is in start request wait status.

    DCMCF_LNST_INIT

The acceptance process for the server-type connection establishment request has ended.

The table below shows the relationship between the status and function availability.

Value of status	Library function availability	
	dc_mcf_tonln()	dc_mcf_tofln()
DCMCF_LNST_LISTEN	N	Y
DCMCF_LNST_RETRY	N	Y
DCMCF_LNST_ONLN_W	Y	Y
DCMCF_LNST_INIT	Y	N

Legend:

Y: Can be used

N: Cannot be used

- resv01

  Fills the area with null characters.

Return values

Return value	Return value (numeric)	Explanation
DCMCFRTN_00000	0	Normal termination.
DCMCFRTN_71001	-12001	The dc_mcf_tlsln() function cannot be accepted because the MCF is under start processing.
DCMCFRTN_71002	-12002	The dc_mcf_tlsln() function cannot be accepted because the MCF is under termination processing.
DCMCFRTN_71004	-12004	A memory shortage occurred during dc_mcf_tlsln() function processing.
DCMCFRTN_71005	-12005	A communication error occurred. For the cause, see the message log file.
DCMCFRTN_71006	-12006	An internal error occurred. For the cause, see the message log file.
DCMCFRTN_71009	-12009	The dc_mcf_tlsln() function is not supported by the applicable MCF communication process.

Return value	Return value (numeric)	Explanation
DCMCFRTN_71010	-12010	Although the request to acquire the acceptance status of the server-type connection establishment request was issued to the MCF communication process, the request was not accepted. For the cause, see the message log file.
DCMCFRTN_72050	-13050	DCNOFLAGS is not set in action.
DCMCFRTN_72052	-13052	NULL is not set in resv01.
DCMCFRTN_72053	-13053	NULL is not set in resv02.
DCMCFRTN_72056	-13056	NULL is set in infcnt.
DCMCFRTN_72057	-13057	NULL is set in inf.
DCMCFRTN_72061	-13061	A value of 0 or smaller or of 240 or greater is specified for mcfid.
DCMCFRTN_72076	-13076	The value 1 is not set in infcnt.

dc_mcf_tofln - Stop accepting server-type connection establishment requests

Format

■ ANSI C, C++

```
#include <dcmcf.h>
int dc_mcf_tofln (DCLONG action, DCLONG mcfid, char *resv01,
                  char *resv02)
```

■ K&R C

```
#include <dcmcf.h>
int dc_mcf_tofln (action, mcfid, resv01, resv02)
DCLONG    action;
DCLONG    mcfid;
char      *resv01;
char      *resv02;
```

Description

The `dc_mcf_tofln()` function stops accepting server-type connection establishment requests.

Arguments whose values are set in the UAP

■ `action`

Set `DCNOFLAGS`.

■ `mcfid`

Specify the MCF communication process identifier of the MCF communication service that is to be processed. The permitted value range is from `1` to `239`.

■ `resv01`, `resv02`

Specify `NULL`.

Return values

Return value	Return value (numeric)	Explanation
`DCMCFRTN_00000`	`0`	Normal termination.
`DCMCFRTN_71001`	`-12001`	The `dc_mcf_tofln()` function cannot be accepted because the MCF is under start processing.

Return value	Return value (numeric)	Explanation
DCMCFRTN_71002	-12002	The dc_mcf_tofln() function cannot be accepted because the MCF is under termination processing.
DCMCFRTN_71004	-12004	A memory shortage occurred during dc_mcf_tofln() function processing.
DCMCFRTN_71005	-12005	A communication error occurred. For the cause, see the message log file.
DCMCFRTN_71006	-12006	An internal error occurred. For the cause, see the message log file.
DCMCFRTN_71009	-12009	The dc_mcf_tofln() function is not supported by the applicable MCF communication process.
DCMCFRTN_71010	-12010	Although the request to stop accepting server-type connection establishment requests was issued to the MCF communication process, the request was not accepted. For the cause, see the message log file.
DCMCFRTN_72050	-13050	DCNOFLAGS is not set in action.
DCMCFRTN_72052	-13052	NULL is not set in resv01.
DCMCFRTN_72053	-13053	NULL is not set in resv02.
DCMCFRTN_72061	-13061	A value of 0 or smaller or of 240 or greater is specified for mcfid.

dc_mcf_tonln - Start accepting server-type connection establishment requests

Format

■ ANSI C, C++

```
#include <dcmcf.h>
int dc_mcf_tonln (DCLONG action, DCLONG mcfid, char *resv01,
                  char *resv02)
```

■ K&R C

```
#include <dcmcf.h>
int dc_mcf_tonln (action, mcfid, resv01, resv02)
DCLONG    action;
DCLONG    mcfid;
char      *resv01;
char      *resv02;
```

Description

The `dc_mcf_tonln()` function starts accepting server-type connection establishment requests.

Arguments whose values are set in the UAP

■ `action`

Set `DCNOFLAGS`.

■ `mcfid`

Specify the MCF communication process identifier of the MCF communication service that is to be processed. The permitted value range is from `1` to `239`.

■ `resv01`, `resv02`

Specify `NULL`.

Return values

Return value	Return value (numeric)	Explanation
`DCMCFRTN_00000`	`0`	Normal termination.
`DCMCFRTN_71001`	`-12001`	The `dc_mcf_tonln()` function cannot be accepted because the MCF is under start processing.

Return value	Return value (numeric)	Explanation
DCMCFRTN_71002	-12002	The dc_mcf_tonln() function cannot be accepted because the MCF is under termination processing.
DCMCFRTN_71004	-12004	A memory shortage occurred during dc_mcf_tonln() function processing.
DCMCFRTN_71005	-12005	A communication error occurred. For the cause, see the message log file.
DCMCFRTN_71006	-12006	An internal error occurred. For the cause, see the message log file.
DCMCFRTN_71009	-12009	The dc_mcf_tonln() function is not supported by the applicable MCF communication process.
DCMCFRTN_71010	-12010	Although the request to start accepting server-type connection establishment requests was issued to the MCF communication process, the request was not accepted. For the cause, see the message log file.
DCMCFRTN_72050	-13050	DCNOFLAGS is not set in action.
DCMCFRTN_72052	-13052	NULL is not set in resv01.
DCMCFRTN_72053	-13053	NULL is not set in resv02.
DCMCFRTN_72061	-13061	A value of 0 or smaller or of 240 or greater is specified for mcfid.

Performance verification trace (dc_prf_~)

This section describes the functions available for the performance verification trace. The functions for the performance verification trace are as follows:

- `dc_prf_get_trace_num` - Report the sequential number for an acquired performance verification trace
- `dc_prf_utrace_put` - Acquire user-specific performance verification traces

The functions (`dc_prf_~`) for the performance verification trace are available on UAPs that run TP1/Server Base or TP1/LiNK. However, you must have installed TP1/Extension 1 before you can use this facility. Note that operation will be unpredictable if you run the facility while TP1/Extension 1 is not installed.

dc_prf_get_trace_num - Report the sequential number for an acquired performance verification trace

Format

- ANSI C, C++

```
#include <dcprf.h>
int dc_prf_get_trace_num(unsigned short *trace_num,
                         DCLONG flags)
```

- K&R C

```
#include <dcprf.h>
int dc_prf_get_trace_num(trace_num, flags)
unsigned short *trace_num;
DCLONG      flags;
```

Description

The function `dc_prf_get_trace_num()` reports the acquired sequential trace number within the process of the latest performance verification trace (`prf` trace) acquired before the function was called. It reports this information to the function call source.

If no performance verification trace has been acquired in the process that called the function `dc_prf_get_trace_num()`, the acquired sequential trace number is 0.

Arguments whose values are set in the UAP

- `trace_num`

  Specify the leading pointer of the area in which you want to set the sequential number for an acquired performance verification trace.

- `flags`

  Specify `DCNOFLAGS`.

Return values

Return value	Return value (numeric)	Explanation
`DC_OK`	`0`	Normal termination.
`DCPRFER_PARAM`	`-4601`	The value specified for an argument is invalid.

dc_prf_utrace_put - Acquire user-specific performance verification traces

Format

- ANSI C, C++

```
#include <dcprf.h>
int dc_prf_utrace_put(unsigned short event_id, unsigned
                      short datalen, char *buffaddr,
                      DCLONG flags)
```

- K&R C

```
#include <dcprf.h>
int dc_prf_utrace_put(event_id, datalen, buffaddr, flags)
unsigned short  event_id;
unsigned short  datalen;
char            *buffaddr;
DCLONG          flags;
```

Description

The function `dc_prf_utrace_put()` acquires a user-specific performance verification trace (`prf` trace).

Arguments whose values are set in the UAP

- `event_id`

  Specify the event ID of the event to be acquired. The range of available event IDs is 0x0001 to 0x0040.

- `datalen`

  Specify the data length of the trace data to be acquired. The specifiable data length is 4 bytes to 256 bytes. The data length must be a multiple of 4 bytes.

- `buffaddr`

  Specify the leading pointer of the buffer holding the trace data to be acquired.

- `flags`

  Specify `DCNOFLAGS`.

Return values

Return value	Return value (numeric)	Explanation
DC_OK	0	Normal termination.
DCPRFER_PARAM	-4601	The value specified for an argument is invalid.

Notes

Even if the function dc_prf_utrace_put() returns the value DC_OK, the trace has not necessarily been properly acquired. This is because data may be lost during trace acquisition processing if multiple processes issue acquisition requests simultaneously because no lock is used.

Remote API facility (dc_rap_~)

This section explains the functions to be used when the user uses remote API facility to manage establishment and release of connections. The functions provided by the remote API facility are as follows:

- `dc_rap_connect` - Establish a connection with a RAP-processing listener
- `dc_rap_disconnect` - Release a connection with a RAP-processing listener

The functions (`dc_rap_~`) provided by the remote API facility can be used in UAPs of TP1/Server Base or TP1/LiNK.

dc_rap_connect - Establish a connection with a RAP-processing listener

Format

- ANSI C, C++

```
#include <dcrap.h>
int dc_rap_connect(char *target_host, DCLONG target_port,
                   DCRAP_SV_ID *sv_id, DCLONG rflags)
```

- K&R C

```
#include <dcrap.h>
int dc_rap_connect(target_host, target_port, sv_id, rflags)
char        *target_host;
DCLONG      target_port;
DCRAP_SV_ID *sv_id;
DCLONG      rflags;
```

Description

The function `dc_rap_connect` establishes a connection between a RAP-processing listener and a RAP-processing client.

The RAP-processing listener with which a connection is to be established is the RAP-processing listener that was activated at `target_port` on `target_host`.

Arguments whose values are set in the UAP

- `target_host` ((1 to 255 alphanumeric characters, periods, or hyphens))

  Specify the host name of the OpenTP1 node on which the RAP-processing listener was activated.

- `target_port` <unsigned integer> ((1 to 65535))

  Specify the port number of the well-known port being used by the RAP-processing listener.

- `rflags`

  Specify `DCNOFLAGS`.

Arguments whose values are returned from OpenTP1

- `sv_id`

  A service ID is returned when the function `dc_rap_connect` terminates normally or `DCRAPER_ALREADY_CONNECT` is returned.

Return values

Return value	Return value (numeric)	Explanation
DC_OK	0	Normal termination. A connection was established with the RAP-processing listener.
DCRAPER_PARAM	-5501	The value specified for the argument is invalid.
DCRAPER_PROTO	-5502	The protocol is invalid. A possible cause is as follows: • The function dc_rpc_open was not called. • Although the rpc_rap_auto_connect operand in the user service definition had been set to Y, the function dc_rap_connect was called. • The -w option was not specified in the dcsvgdef definition command in the user service network definition.
DCRAPER_NOMEMORY	-5503	The memory became insufficient.
DCRAPER_MAX_CONNECTION	-5517	The specified value exceeds the maximum number of dc_rap_connect functions which can be called from a single process.
DCRAPER_NETDOWN	-5505	A network error occurred during communication with the RAP-processing listener.
DCRAPER_TIMEDOUT	-5506	A timeout occurred during communication with the RAP-processing listener.
DCRAPER_NOSOCKET	-5507	The number of sockets became insufficient.
DCRAPER_NOHOSTNAME	-5508	The host name cannot be resolved.
DCRAPER_SHUTDOWN	-5521	The RAP-processing listener is being terminated.
DCRAPER_NOCONTINUE	-5522	An error which prevents continuation of processing occurred. Possible causes of the error are as follows: • An unexpected message was received. • A message was received unexpectedly from a remote system.
DCRAPER_SYSCALL	-5523	An unexpected error occurred during system call.
DCRAPER_UNKNOWN_NODE	-5531	An attempt was made to establish a connection with a RAP-processing listener which is on an unconnected network.
DCRAPER_NOMEMORY_SV	-5520	The memory became insufficient on the RAP-processing listener or RAP-processing server.

Return value	Return value (numeric)	Explanation
DCRAPER_TIMEOUT_SV	-5532	A connection could not be established within the message exchange monitoring time specified in the rap_watch_time operand of the RAP-processing listener service definition.
DCRAPER_PANIC_SV	-5533	A system error occurred in the RAP-processing listener.
DCRAPER_MAX_CONNECTION_SV	-5534	The specified value exceeds the maximum number of requests which can be accepted for connection with a RAP-processing client that is managed by a RAP-processing listener.
DCRAPER_NOSERVICE	-5528	The RAP-processing listener is being started or terminated.
DCRAPER_ALREADY_CONNECT	-5529	A connection has already been established with the RAP-processing listener.

Note

If the function dc_rap_connect returns with an error (returns with a value other than DCRAPER_ALREADY_CONNECT), connection was not established with the RAP-processing listener.

The error code acquired by the UAP trace is as follows:

0: No error

1: The function dc_rpc_open() was not called.

3: The value specified for the host name contains an error.

4: The value specified for the port number contains an error.

5: An area for storing the service ID was not specified.

6: The dc_rap_connect() function was called while the value Y was specified in the rpc_rap_auto_connect operand in the user service definition. Alternatively, the user service network definition has not been defined.

dc_rap_disconnect - Release a connection with a RAP-processing listener

Format

- ANSI C, C++

```
#include <dcrap.h>
int dc_rap_disconnect(DCRAP_SV_ID sv_id, DCLONG rflags)
```

- K&R C

```
#include <dcrap.h>
int dc_rap_connect(sv_id, rflags)
DCRAP_SV_ID sv_id;
DCLONG      rflags;
```

Description

The function `dc_rap_disconnect` releases a connection established between a RAP-processing listener and a RAP-processing client.

Arguments whose values are set in the UAP

- `sv_id`

  Specify the service ID that was received for the function `dc_rap_connect`.

- `rflags`

  Specify `DCNOFLAGS`.

Return values

Return value	Return value (numeric)	Explanation
`DC_OK`	`0`	Normal termination. The connection with the RAP-processing listener was released.
`DCRAPER_PARAM`	`-5501`	The argument is invalid. Possible causes are as follows: • The service ID differs from the service ID received by the function `dc_rap_connect`.

Return value	Return value (numeric)	Explanation
DCRAPER_PROTO	-5502	The protocol is invalid. Possible causes of the error are as follows: • The function dc_rpc_open was not called. • Although the rpc_rap_auto_connect operand in the user service definition had been set to Y, the function dc_rap_disconnect was called. • The -w option was not specified in the dcsvgdef definition command in the user service network definition.
DCRAPER_NOMEMORY	-5503	The memory became insufficient.
DCRAPER_NETDOWN	-5505	A network error occurred during communication with the RAP-processing listener.
DCRAPER_TIMEDOUT	-5506	A timeout occurred during communication with the RAP-processing listener.
DCRAPER_SHUTDOWN	-5521	The RAP-processing listener is being terminated.
DCRAPER_NOCONTINUE	-5522	An error which prevents continuation of processing occurred. Possible causes of the error are as follows: • An unexpected message was received. • A message was received unexpectedly from a remote system.
DCRAPER_SYSCALL	-5523	An unexpected error occurred during system call.

Note

If the function dc_rap_disconnect returns with an error (returns with a value other than DCRAPER_PARAM or DCRAPER_PROTO), the connection with the RAP-processing listener was released. The error code acquired by the UAP trace is as follows:

0: No error

1: The function dc_rpc_open() was not called.

3: The dc_rap_disconnect() function was called while the value Y was specified in the rpc_rap_auto_connect operand in the user service definition.

Remote procedure call (dc_rpc_~)

This section gives the syntax and other information of the following OpenTP1 remote procedure call functions which are used for client-server communication.

- dc_rpc_call - Request a remote service
- dc_rpc_call_to - Invoke a remote service with a communication destination specified
- DCRPC_BINDTBL_SET, DCRPC_DIRECT_SCHEDULE - Create the DCRPC_BINDING_TBL structure
- dc_rpc_close - Terminate an application program
- dc_rpc_cltsend - Report data to CUP unidirectionally
- dc_rpc_discard_further_replies - Reject the receiving of processing results
- dc_rpc_discard_specific_reply - Reject acceptance of particular processing results
- dc_rpc_get_callers_address - Acquire the node address of a client UAP
- dc_rpc_get_error_descriptor - Acquire the descriptor of an asynchronous response-type RPC request which has encountered an error
- dc_rpc_get_gateway_address - Acquire the node address of a gateway
- dc_rpc_get_service_prio - Reference the schedule priority of a service request
- dc_rpc_get_watch_time - Reference the service response waiting inter - val
- dc_rpc_mainloop - Start an SPP service
- dc_rpc_open - Start an application program
- dc_rpc_poll_any_replies - Receive processing results in asynchronous mode
- dc_rpc_service_retry - Retry a service function
- dc_rpc_set_service_prio - Set a schedule priority of a service request
- dc_rpc_set_watch_time - Update a service response waiting interval

The functions for remote procedure call (dc_rpc_~) can be used in UAPs of both TP1/Server Base and TP1/LiNK.

dc_rpc_call - Request a remote service

Format

- ANSI C, C++

```
#include <dcrpc.h>
int  dc_rpc_call (char *group, char *service, char *in,
                  DCULONG *in_len, char *out,
                  DCULONG *out_len, DCLONG flags)
```

- K&R C

```
#include <dcrpc.h>
int  dc_rpc_call (group, service, in, in_len, out, out_len,
                  flags)
char       *group;
char       *service;
char       *in;
DCULONG    *in_len;
char       *out;
DCULONG    *out_len;
DCLONG     flags;
```

Description

The function `dc_rpc_call()` requests an SPP service. This function can be called without consideration of the node containing the requesting service.

Specify a service group name and service name as arguments of the function `dc_rpc_call()` to request a service. A service request is addressed to the service function corresponding to the specified names.

A UAP which calls the function `dc_rpc_call()` can be used regardless of whether it has been executed as a transaction. When a service is requested by the function `dc_rpc_call()` from the process which has been executed as a transaction, the requested service process runs as a transaction branch.

Before this function can be used, the OpenTP1 at the node containing the server UAP to which the service request is addressed must be active.

Receiving a signal while waiting for a response after execution of the function `dc_rpc_call()` does not cause the function to be returned.

The following items are described after the list of return values. See each description For details on the function `dc_rpc_call()`.

(1) Arguments of the function `dc_rpc_call()`

(2) Error cases of the function `dc_rpc_call()`

(3) Timing when the function dc_rpc_call() results in error

(4) Specification for reexecuting the service request if the function dc_rpc_call() results in error

(5) When a priority is given to a service request

(6) Difference between return values DCRPCER_NO_SUCH_SERVICE_GROUP and DCRPCER_NET_DOWN

(7) Specification for returning the value DCRPCER_SERVICE_TERMINATED

(8) Relationship between return values and synchronization point processing

(9) Notes on requesting a service

(10) When a service is requested with domain qualification

Arguments whose values are set in the UAP

- group

  Specify the SPP service group name with an ASCII character string of up to 31 bytes. End the character string with a null character. The null character is not counted in the length of the character string.

  When requesting a service with domain qualification, specify the service group name suffixed by an at mark (@) and the DNS domain name, and end the character string with a null character.

- service

  Specify the SPP service name with an ASCII character string of up to 31 bytes. End the character string with a null character. The null character is not counted in the length of the character string.

- in

  Specify the input parameter of the service.

- in_len

  Specify the input parameter length of the service within the range from 1 to DCRPC_MAX_MESSAGE_SIZE#. DCRPC_MAX_MESSAGE_SIZE is defined in dcrpc.h.

  #: If you used the rpc_max_message_size operand, the value of this data area is the value specified in the rpc_max_message_size operand and not the value of DCRPC_MAX_MESSAGE_SIZE (1 megabyte).

- out

  Specify the area for the response from the service function. This area will receive the response from the service function.

- out_len

  Specify the length of the response from the service function within the range from 1 to DCRPC_MAX_MESSAGE_SIZE#. DCRPC_MAX_MESSAGE_SIZE is defined in dcrpc.h.

  #: If you used the rpc_max_message_size operand, the value of this data area is the value specified in the rpc_max_message_size operand and not the value of DCRPC_MAX_MESSAGE_SIZE (1 megabyte).

  Even if the RPC is the non response-type, you must specify the address of the area for which the length of the response from the service is specified. If it is, the length of the response from the service must be 0.

- flags

  Specify the RPC mode and option in the following format:

```
{DCNOFLAGS|DCRPC_NOWAIT|DCRPC_NOREPLY|DCRPC_CHAINED}
[|DCRPC_TPNOTRAN][|DCRPC_DOMAIN]
```

  DCNOFLAGS

  Synchronous response-type RPC

  DCRPC_NOWAIT

  Asynchronous response-type RPC

  DCRPC_NOREPLY

  Nonresponse-type RPC

  DCRPC_CHAINED

  Chained RPC

  DCRPC_TPNOTRAN

  Specify this option not to handle the requested processing as a transaction. When DCRPC_TPNOTRAN is specified, the processing of the service function is not handled as a transaction even if the service is requested from the transaction.

  DCRPC_DOMAIN

  Specify this option when the service group name is specified with domain qualification. An RPC with domain qualification cannot be a transaction branch. Therefore, specify this option together with DCRPC_TPNOTRAN whenever the function dc_rpc_call() is used from the transaction.

  Specify DCRPC_TPNOTRAN and/or DCRPC_DOMAIN together with the RPC mode.

  Example 1:

  When a nontransaction service is requested by using a synchronous response-type

RPC, specify for flags as follows:

```
DCNOFLAGS|DCRPC_TPNOTRAN
```

Example 2:

When a service is requested by using a synchronous response-type RPC with domain qualification from the transaction, specify for flags as follows:

```
DCNOFLAGS|DCRPC_TPNOTRAN|DCRPC_DOMAIN
```

Arguments whose values are returned from server UAP

- `out`

  The response set by the service function is returned.

- `out_len`

  The length of the response set by the service function is returned.

Return values

The following return values are returned from the OpenTP1, not from the service function.

Return value	Return value (numeric)	Explanation
`0` or positive integer		Normal termination. In the case of asynchronous response-type RPC, the positive integer is the descriptor.
`DCRPCER_INVALID_ARGS`	`-301`	The value specified for the argument is invalid.
`DCRPCER_PROTO`	`-302`	The function `dc_rpc_open()` was not called.
`DCRPCER_NO_BUFS`	`-304`	A memory shortage occurred. Or, a service request was not accepted because a space shortage occurred in the message storage buffer pool (`message_store_buflen` operand) of the SPP to which the service was requested. If necessary, revise the `message_store_buflen` operand in the user service default definition or in the user service definition of the SPP to which the service was requested.
`DCRPCER_NET_DOWN`	`-306`	A communication failure occurred. Check if a network failure has occurred.

Return value	Return value (numeric)	Explanation
DCRPCER_TIMED_OUT	-307	The response wait time in the dc_rpc_call() function has elapsed. If necessary, revise the response wait time specified in the dc_rpc_call() function (watch_time operand and arguments in the dc_rpc_set_watch_time() function).
		The SPP to which the service was requested terminated abnormally during execution of a service function. Check the cause of abnormal termination of the SPP to which the service was requested.
DCRPCER_MESSAGE_TOO_BIG	-308	The input parameter length specified in in_len exceeded the maximum. If necessary, revise the in_len setting.
DCRPCER_REPLY_TOO_BIG	-309	The length of the response (out_len) set in the service function of the SPP to which the service was requested exceeded the response length (out_len) in the dc_rpc_call() function. If necessary, revise the response length (out_len) set in the service function of the SPP to which the service was requested.
DCRPCER_NO_SUCH_SERVICE_GROUP	-310	The service group name set in group is invalid, or the SPP to which the service was requested with the service group set in group was not running. If necessary, revise the group setting, or start the SPP to which the service was requested with the service group set in group.
DCRPCER_NO_SUCH_SERVICE	-311	The service name set in service is invalid, or the service name set in service by the SPP to which the service was requested has not been specified in the service operand in the user service definition file. If necessary, revise the service setting, or specify the service name set in service also in the service operand for the SPP to which the service was requested.
DCRPCER_SERVICE_CLOSED	-312	The SPP to which the service was requested with the service group set in group is under server shutdown or service shutdown status. Check the cause of the shutdown, and then release the SPP from shutdown status.
DCRPCER_SERVICE_TERMINATING	-313	The SPP to which the service was requested is under termination processing.

Return value	Return value (numeric)	Explanation
DCRPCER_SERVICE_NOT_UP	-314	The SPP to which the service was requested with the service group set in group is not running, or a communication failure might have occurred during the service request send processing. Start the SPP to which the service was requested with the service group set in group. If the SPP is already running, check to see if a network failure has occurred.
		While 0 was specified for the service request response time (watch_time operand and an argument in the dc_rpc_set_watch_time() function), the SPP to which the service was requested terminated abnormally during execution of a service function. Check the cause of abnormal termination of the SPP to which the service was requested.
DCRPCER_OLTF_NOT_UP	-315	OpenTP1 for the SPP to which the service was requested is not running. OpenTP1 might be under termination processing or a communication failure might have occurred during the service request send processing. Start OpenTP1 for the SPP to which the service was requested, or check for a network failure.
DCRPCER_SYSERR_AT_SERVER	-316	A system error (internal conflict) occurred in the SPP to which the service was requested.
DCRPCER_NO_BUFS_AT_SERVER	-317	A memory shortage occurred in the SPP to which the service was requested.
DCRPCER_SYSERR	-318	A system error (internal conflict) occurred in the UAP that requested the service.
DCRPCER_INVALID_REPLY	-319	The response length (out_len) set by a service function of the SPP to which the service was requested is outside the range from 1 to the value defined in DCRPC_MAX_MESSAGE_SIZE.# If necessary, revise the response length (out_len) in the service function of the SPP to which the service was requested.
DCRPCER_OLTF_INITIALIZING	-320	OpenTP1 for the SPP to which the service was requested is under start processing.

Return value	Return value (numeric)	Explanation
DCRPCER_NO_BUFS_RB	-323	A memory shortage occurred in the UAP that is requesting the service or the SPP to which the service was requested. When this value is returned, the transaction branch rolls back. Check whether unneeded memory was allocated by the UAP that is requesting the service, or by the SPP to which the service was requested.
DCRPCER_SYSERR_RB	-324	A system error (internal conflict) occurred in the UAP that requested the service. When this value is returned, the transaction branch rolls back.
DCRPCER_SYSERR_AT_SERVER_RB	-325	A system error (internal conflict) occurred in the SPP to which the service was requested. When this value is returned, the transaction branch rolls back.
DCRPCER_REPLY_TOO_BIG_RB	-326	The response length (out_len) set in the service function of the SPP to which the service was requested exceeded the response length (out_len) in the dc_rpc_call() function. When this value is returned, the transaction branch rolls back. If necessary, revise the response length (out_len) set in the service function of the SPP to which the service was requested.
DCRPCER_TRNCHK	-327	When the inter-node load-balancing facility and the extended internode load-balancing facility are used, the transaction attributes (atomic_update operand) do not match among the SPPs with the same service group name to which the service was requested. Another possibility is that the inter-node load-balancing facility and the extended internode load-balancing facility cannot be used because the version of OpenTP1 at the node to which loads are to be distributed is earlier than that of the OpenTP1 for the UAP that is requesting the service. This value is returned only when the service request is issued to an SPP that uses the inter-node load-balancing facility and the extended internode load-balancing facility. If necessary, revise the transaction attribute (atomic_update operand) of the SPP that uses the inter-node load-balancing facility and the extended internode load-balancing facility, or revise if necessary the version of OpenTP1.

Return value	Return value (numeric)	Explanation
		The dcsvgdef definition command was used to issue a service request to a user server with the non-transaction attribute (the atomic_update operand is N in the user service definition or the jnl_fileless_option operand is Y in the system common definition), but a disjunction with DCRPC_TPNOTRAN was not specified in the flags argument of the dc_rpc_call() function. If necessary, revise the dcsvgdef definition command or the flags argument of the dc_rpc_call() function.
DCRPCER_NO_SUCH_DOMAIN	-328	The domain name of the service group name with the domain qualification in group is invalid. If necessary, revise the domain name.
DCRPCER_NO_PORT	-329	A service was requested with a domain qualification in group, but the port number of the domain representative schedule service was not found. If necessary, revise the domain_masters_port operand setting in the system common definition and the port number setting for the domain representative schedule service in /etc/services.
DCRPCER_SERVER_BUSY	-356	The SPP to which the service was requested (on a server that receives requests through a socket) cannot receive the service request. If necessary, revise the max_socket_msg and max_socket_msglen operands in the user service definition or the user service default definition for the SPP to which the service was requested.
DCRPCER_TESTMODE	-366	When the online tester was being used, a service request was issued from a UAP in the test mode to an SPP in the nontest mode or from a UAP in the nontest mode to an SPP in the test mode. If necessary, revise the UAP's test mode setting.
DCRPCER_NOT_TRN_EXTEND	-367	The dc_rpc_call() function with DCRPC_TPNOTRAN set in flags was called to request a service after a chained RPC with the transaction attribute was executed.

Return value	Return value (numeric)	Explanation
DCRPCER_SECCHK	-370	The SPP to which the service was requested is protected by the security facility. The UAP that requested the service by executing the dc_rpc_call() function does not have permission to access the SPP to which the service was requested. If necessary, revise the access permissions for the SPP to which the service was requested.
DCRPCER_TRNCHK_EXTEND	-372	The transaction branch cannot be started because it exceeds the maximum number of transaction branches that can be activated concurrently in the OpenTP1 for the SPP to which the service was requested. If necessary, revise the setting in the trn_tran_process_count operand in the transaction service definition.
		The transaction branch cannot be started because it exceeds the maximum number of child transaction branches that can be activated from one transaction branch by the UAP that is requesting the service. If necessary, revise the setting in the trn_max_subordinate_count operand in the transaction service definition.
		DCRPC_TPNOTRAN is not specified for flags when a service with domain qualification specified in a transaction is requested.
		Transaction branching cannot start because the SPP to which the service was requested encountered a resource manager (RM) error. Eliminate the cause of the resource manager (RM) error, and then re-execute the function.
		In the System Environment window of TP1/LiNK, the **Transaction Facility** item is not set to **Yes**. If necessary, revise the **Transaction Facility** setting in the System Environment window of TP1/LiNK.

Return value	Return value (numeric)	Explanation
DCRPCER_SERVICE_TERMINATED	-378	The SPP to which the service was requested terminated abnormally during service function execution. If necessary, revise the service function processing of the SPP to which the service was requested. This value is returned only for a UAP that was requesting a service for which 00000001 was specified in the rpc_extend_function operand in the user service definition. If 0000000 is specified in the rpc_extend_function operand or if the operand is omitted, DCRPCER_TIMED_OUT or DCRPCER_SERVICE_NOT_UP is returned rather than this value.

#: If you used the rpc_max_message_size operand, the value of this data area is the value specified in the rpc_max_message_size operand and not the value of DCRPC_MAX_MESSAGE_SIZE (1 megabyte).

(1) Arguments of the function dc_rpc_call()

Arguments of the function dc_rpc_call() are explained below.

■ Values passed to server UAP

Allocate an area (out) for the response from the service function before requesting a service. The client UAP sets the following values in the function dc_rpc_call().

- Input parameter (in)
- Input parameter length (in_len)
- Response length (out_len)

The input parameter, input parameter length, and response length values which are set in the function dc_rpc_call() of the client UAP are passed to the service function as is. Change the notation of character codes or digits in the processing of the client UAP or requested service function if required. If a service request is addressed to the service function which does not return any response, the specified response length is ignored.

The maximum values of input parameter length and response length are declared as DCRPC_MAX_MESSAGE_SIZE# in the header file dcrpc.h. See the contents of dcrpc.h to confirm the maximum values.

#: If you used the rpc_max_message_size operand, the value of this data area is the value specified in the rpc_max_message_size operand and not the value of DCRPC_MAX_MESSAGE_SIZE (1 megabyte).

- Values returned from server UAP

  When the service function terminates and response is returned, the following values can be referenced:

  - Response from service function (out)
  - Length of response from service function (out_len)

  The value of out_len is the length of the response which is actually returned from the service function. The values of out and out_len can be referenced in the following cases depending on the RPC mode:

  - In the case of synchronous response-type RPC and chained RPC

    The values of out and out_len can be referenced when the function dc_rpc_call() returns.

  - In the case of asynchronous response-type RPC

    The value of out can be referenced when the function dc_rpc_poll_any_replies() which receives the response returns. The value of out_len cannot be referenced.

  - In the case of nonresponse-type RPC

    The values of out and out_len cannot be referenced.

  If the function dc_rpc_call() or dc_rpc_poll_any_replies() returns with an error, the values of out and out_len cannot be referenced.

  If the returned response is larger than the response area acquired by the client UAP, the function returns with an error, giving the return value DCRPCER_REPLY_TOO_BIG.

- Value specified for flags

  The value specified for flags and the execution result of the function dc_rpc_call() are explained below.

  - Synchronous response-type RPC (when DCNOFLAGS is specified for flags)

    The function dc_rpc_call() will not return until a response returns or a communication error occurs.

  - Asynchronous response-type RPC (when DCRPC_NOWAIT is specified for flags)

    The function dc_rpc_call() will return immediately. The response can be referenced after the response is received asynchronously in the function dc_rpc_poll_any_replies(). Do not free the response storage area (out) until the asynchronous response-type RPC is terminated due to one of the following causes:

  - A response is received by the function dc_rpc_poll_any_replies()

- The receiving of responses is rejected by the function `dc_rpc_discard_further_replies()`
- Commitment or rollback is performed when a service is requested from a transaction.

  When an asynchronous response-type RPC is used in a transaction, receive responses by using the function `dc_rpc_poll_any_replies()` before performing the synchronization point processing (commitment or rollback). No response can be received by the function `dc_rpc_poll_any_replies()` after the synchronization point processing. To designate a specific response received by the function `dc_rpc_poll_any_replies()`, specify the positive integer (descriptor), which is returned by the function `dc_rpc_call()`, as the argument of the function `dc_rpc_poll_any_replies()`. Thus, hold the return value of the function `dc_rpc_call()` to designate a specific response received.

  To receive responses after the synchronization point processing while in non-transaction processing, specify the corresponding option in the `rpc_extend_function` operand of the system service definition.

  For details about `rpc_extend_function`, see the manual *OpenTP1 System Definition*.
- Nonresponse-type RPC (when `DCRPC_NOREPLY` is specified for `flags`)

  The function `dc_rpc_call()` will return immediately without waiting for completion of the service function processing. The service function is treated as a function which does not return any response. Therefore, the UAP requesting a service cannot determine whether the service function has been performed. With this specification, the response (`out`) and its length (`out_len`) cannot be referenced.
- Chained RPC (when `DCRPC_CHAINED` is specified for `flags`)

  The function `dc_rpc_call()` will not return until a response is returned or a communication error occurs. If two or more services belonging to the same service group in chained RPCs are requested, the subsequent services can be handled in the same process as for the service requested first.

There are the following restrictions on the use of chained RPCs:

1. The shutdown state of the user server or service cannot be detected by the second and subsequent calls of the function `dc_rpc_call()`.
2. The entire user server enters in shutdown state if an error occurs during the service function processing of the second and subsequent calls of the function `dc_rpc_call()`. Services do not enter in shutdown state individually.

(2) Error cases of the function dc_rpc_call()

Reasons why the function `dc_rpc_call()` returns with an error are explained below.

- If the OpenTP1 at the node containing the server UAP is not active

  If the OpenTP1 to which the service request is addressed is not active, the function `dc_rpc_call()` returns with an error, giving one of the following return values:

  - `DCRPCER_NET_DOWN`
  - `DCRPCER_SERVICE_NOT_UP`
  - `DCRPCER_OLTF_NOT_UP`
  - `DCRPCER_OLTF_INITIALIZING`

- If the server UAP is not active

  When the server UAP is a multiserver, the service request is dealt with a new process which is activated by the OpenTP1 even if the server UAP is being terminated abnormally or being partially recovered. However, the function `dc_rpc_call()` returns with an error in the following cases:

  1. No service request can be addressed to the SPP in shutdown state. If the service group is shut down, the function `dc_rpc_call()` returns with an error, giving the return value `DCRPCER_SERVICE_CLOSED`.
  2. If the SPP is being terminated or has been terminated by the stop command for the user server (`dcsvstop` command) or for OpenTP1 (`dcstop` command), the `dc_rpc_call()` function returns with an error and sets one of the following status code values:

     - `DCRPCER_SERVICE_TERMINATING`
     - `DCRPCER_SERVICE_CLOSED`
     - `DCRPCER_NO_SUCH_SERVICE_GROUP`

     The value that is returned depends on the timing of calling the `dc_rpc_call()` function.
  3. If the OpenTP1 is being started, the function `dc_rpc_call()` returns with an error, giving the return value `DCRPCER_OLTF_INITIALIZING`. In this case, a service may be requested normally after activation of the server UAP or OpenTP1 is completed. Since activation of the OpenTP1 is completed when a message log with the message ID `KFCA01809-I` is output, request a service again after this message appears.

- When a service is requested in the environment for the internode load-balancing facility and the extended internode load-balancing facility

  In the environment for the internode load-balancing facility and the extended internode load-balancing facility, if the schedule of the applicable service is closed, OpenTP1 automatically transfers a service request to another node. However, the function `dc_rpc_call` returns `DCRPCER_TRNCHK`, and control is returned due to an error

under either of the following conditions:

1. For transaction processing, the transaction attribute of the service on the transfer destination node does not match the closed service.
2. The version of the OpenTP1 on the transfer destination node is earlier than that of the node for the OpenTP1 that requested the service.

When control is returned as a result of the foregoing error, take the following actions:

1. Force the transaction attributes of the SPPs making up the internode load-balancing facility and the extended internode load-balancing facility to match.
2. Force the OpenTP1 versions making up the internode load-balancing facility and the extended internode load-balancing facility to match.

- When a service request is addressed to the server that receives requests from socket

  The server that receives requests from socket controls message congestion according to the specified values for `max_socket_msg` and `max_socket_msglen` in the user service definition. It is probable that service requests cannot be accepted if a message exceeds the defined value. In this case, the function `dc_rpc_call()` returns with an error, giving the return value `DCRPCER_SERVER_BUSY`. If this value is returned, the client UAP can sometimes reissue the service request successfully after waiting for a while.

- When a chained RPC is used

  If the function `dc_rpc_call()` which is not a transaction is called from the UAP using a chained RPC which is processed as a transaction to the same server UAP, the function `dc_rpc_call()` returns with an error, giving the return value `DCRPCER_NOT_TRN_EXTEND`.

- When the online tester is used

  If the online tester is in use and the function `dc_rpc_call()` is called from a UAP in test mode to a UAP in nontest mode or vice versa, the function `dc_rpc_call()` returns with an error, giving the return value `DCRPCER_TESTMODE`.

- When the security facility is used

  If the desired service is protected with the security facility when the function `dc_rpc_call()` is called and the client UAP which called the function does not have the access permission for the SPP, the function `dc_rpc_call()` returns with an error, giving the return value `DCRPCER_SECCHK`.

(3) Timing when the function dc_rpc_call() results in error

The following explains the timing when an error is returned to the client UAP if the SPP to which the service request is addressed terminates abnormally.

- Synchronous response-type RPC or chained RPC (when DCNOFLAGS or DCRPC_CHAINED is specified for flags)

  If an SPP which executes a service terminates abnormally before completion of the processing, the function dc_rpc_call() returns with an error, giving the return value DCRPCER_TIMED_OUT. If an infinite period of time is specified in the watch_time operand in the user service definition of the client UAP, the function returns with an error, giving the return value DCRPCER_SERVICE_NOT_UP.

- Asynchronous response-type RPC (when DCRPC_NOWAIT is specified for flags)

  If an SPP which executes a service terminates abnormally before completion of the processing, the function dc_rpc_poll_any_replies() returns with an error, giving the return value DCRPCER_TIMED_OUT. If an infinite period of time is specified in the watch_time operand in the user service definition of the client UAP, the function returns with an error, giving the return value DCRPCER_SERVICE_NOT_UP.

- Nonresponse-type RPC (when DCRPC_NOREPLY is specified for flags)

  The client UAP cannot detect abnormal termination of server UAP.

■ When the function dc_rpc_call() results in error due to time monitoring of the client UAP

In the following cases, the function dc_rpc_call() returns with an error, giving the return value DCRPCER_TIMED_OUT, after the time specified in the watch_time operand in the user service definition of the client UAP has elapsed:

- The entire OpenTP1 at the node containing the SPP terminates abnormally.
- An error occurs before the server UAP receives service request data or before the client UAP receives the result after the server UAP processing is completed.

(4) Specification for reexecuting the service request if the function dc_rpc_call() results in error

Even if the OpenTP1 to which the service request is issued is not active because it is being started or is engaged in system switching, you can have the OpenTP1 re-execute the requested search and service request transmission without treating the dc_rpc_call() function processing as an error.

To re-execute the requested search and service request transmission, specify Y in the rpc_retry operand in the system common definition. You use the rpc_retry_count and rpc_retry_interval operands to specify the re-execution count and re-execution interval, respectively, for a requested search and service request transmission. If this count value exceeds the re-execution count value specified in the system common definition, the dc_rpc_call() function returns with

an error and sets one of the following status code values:

- DCRPCER_INVALID_ARGS
- DCRPCER_NET_DOWN
- DCRPCER_SERVICE_NOT_UP
- DCRPCER_NO_SUCH_SERVICE_GROUP
- DCRPCER_OLTF_NOT_UP
- DCRPCER_OLTF_INITIALIZING

(5) When a priority is given to a service request

To specify a schedule priority for a service request, call the function dc_rpc_set_service_prio() immediately before the function dc_rpc_call(). If no schedule priority is specified, the priority of the service request is determined according to the default interpretation of the schedule service.

(6) Difference between return values DCRPCER_NO_SUCH_SERVICE_GROUP and DCRPCER_NET_DOWN

These return values are returned if the user server corresponding to the service group name is not found.

- DCRPCER_NO_SUCH_SERVICE_GROUP

  Indicates the user server is not found after searching all nodes specified for all_node in the system common definition.

- DCRPCER_NET_DOWN

  Indicates a communication error occurred on one or more nodes specified for all_node during the search. This return value may indicate the corresponding OpenTP1 system is not found.

(7) Specification for returning the value DCRPCER_SERVICE_TERMINATED

You may want to determine whether the SPP that requested a service terminated abnormally before completion of processing based on a returned value other than DCRPCER_TIMED_OUT or DCRPCER_SERVICE_NOT_UP. If so, specify 00000001 in the rpc_extend_function operand of the user service definition. This specification returns DCRPCER_SERVICE_TERMINATED if the above error occurs. If 00000000 is specified in the rpc_extend_function operand, or the operand is omitted, DCRPCER_TIMED_OUT or DCRPCER_SERVICE _NOT_UP is returned rather than DCRPCER_SERVICE_TERMINATED.

(8) Relationship between error return values and synchronization point processing

The relationship between return values of the function dc_rpc_call() and synchronization point processing (commitment and rollback) is explained below. The description applies to the service request which is a transaction, rather than the service request which is not a transaction (including the case when DCRPC_TPNOTRAN is specified for flags).

- When commitment is performed even though the function dc_rpc_call() returns with an error

  The return value DCRPCER_TIMED_OUT may be returned due to abnormal termination of the service function which the service request is addressed, a node error, or network error. However, when the client UAP is not a transaction, the service function to which the service request is addressed may terminate normally and database may be updated.

- Error return values which require rollback processing

  If the function dc_rpc_call() called from a transaction returns with an error, some return values always require rollback processing for the transaction (the server UAP enters rollback_only state). In this case, rollback processing is always performed even if either of the commitment function or rollback function is used. The following return values of the function dc_rpc_call() always require rollback processing for the transaction:

  - DCRPCER_INVALID_REPLY
  - DCRPCER_NO_BUFS_AT_SERVER
  - DCRPCER_NO_SUCH_SERVICE
  - DCRPCER_REPLY_TOO_BIG_RB

(9) Notes on requesting a service

1. Define the service group name and service name at server UAP environment setup. These names are set in the function dc_rpc_call(). If a service is requested while invalid service group name or service name is set in the function dc_rpc_call(), the function returns with an error, giving the return value DCRPCER_NO_SUCH_SERVICE_GROUP or DCRPCER_NO_SUCH_SERVICE. If the service function does not return response, the function dc_rpc_call() does not return with an error, giving the return value DCRPCER_NO_SUCH_SERVICE.
2. The process of the server UAP is different from that of the client UAP. Therefore, the following matters are different from ordinary function calls and procedure calls:
   - Attributes (such as environment variables, schedule priority (nice value)) which are given to the process of the client UAP by the OS are not passed on

to the server UAP.

- Environment settings (such as existence of specification of transaction attribute, time limit of transaction branch, schedule priority) of the OpenTP1 specified at the node of the client UAP are not passed on to the OpenTP1 of the server UAP.

3. Do not specify the same buffer area for the input parameter (`in`) and the response from the service function (`out`).
4. If `DCRPC_NOREPLY` is specified for `flags`, the following return values will not return:

   - Errors which never occur

     `DCRPCER_REPLY_TOO_BIG`

     `DCRPCER_INVALID_REPLY`

   - Errors which cannot be detected even though they could occur

     `DCRPCER_NO_SUCH_SERVICE`

     `DCRPCER_SERVICE_CLOSED`

     `DCRPCER_SERVICE_TERMINATING`

     `DCRPCER_SYSERR_AT_SERVER`

     `DCRPCER_NO_BUFS_AT_SERVER`

     `DCRPCER_OLTF_INITIALIZING`

     `DCRPCER_SECCHK`

   In addition, OpenTP1 does not output a message when an error occurs. If errors must be detected, consider specifying `DCNOFLAGS` for `flags` (synchronous-response-type RPC).
5. When a service group is requested by the function `dc_rpc_call()` from a transaction, an SPP is occupied until the transaction terminates. When the same service is requested more than once by the function `dc_rpc_call()` from one transaction, do the following:

   - Re-estimate the values specified for the `balance_count` operand and `parallel_count` operand in the user service definition according to the number of usages.
   - Request a service by using chained RPCs so that the number of processes will not increase.

   If the values specified for the `balance_count` operand and `parallel_count` operand are incorrect, the transaction will shut down abnormally and a deadlock may occur.

6. When an asynchronous response-type RPC is used, the server UAP may be occupied until the function `dc_rpc_poll_any_replies()` receives all asynchronous responses or the function `dc_rpc_discard_further_replies()` rejects the receiving of asynchronous responses. This may occur regardless of whether it is a transaction or not. Increase the number of resident processes according to how many times an asynchronous response-type RPC is used.

   An asynchronous response-type RPC requires many resources in addition to occupying the server UAP. To prevent responses from degrading performance of UAP processing and activation of unnecessary SPPs, ensure that responses are received or the receiving of responses is rejected after the function `dc_rpc_call()` of an asynchronous response-type RPC is used.

7. When a response is received after an asynchronous response-type RPC is used twice or more consecutively, specify a separate response storage area (`out`) for each. If the same area is specified, a correct response cannot be received since the second and succeeding responses override the area.

8. The server UAP (SPP) that requested a service using an asynchronous response-type RPC sends a response soon after the service function is executed, regardless of whether the process that executed the asynchronous response-type RPC issued the function `dc_rpc_poll_any_replies`. If the same asynchronous response-type RPC is executed numerous times simultaneously without the function `dc_rpc_poll_any_replies` being issued, the response sent by the SPP may stay in the TCP/IP buffer and the SPP may fail to send a response. If the SPP fails to send a response, no response can be received from the SPP even if the source of the asynchronous response-type RPC issues the function `dc_rpc_poll_any_replies`.

9. If a large number of asynchronous response-type RPCs or non-response type RPCs having the transaction attribute are executed, messages about transactions sent by the SPP can no longer be received. In this case, the transactions may roll back.

(10) When a service is requested with domain qualification

Specifying a service group name with domain qualification enables requesting an OpenTP1 service in the DNS domain. Specify the service group name suffixed by an at mark (@) and the DNS domain name for domain qualification.

- Notes on requesting a service with domain qualification

  1. To request a service with domain qualification, specify `DCRPC_DOMAIN` for `flags` of the function `dc_rpc_call()`. If the service group name with domain qualification is specified without `DCRPC_DOMAIN`, the function `dc_rpc_call()` returns with an error, giving the return value `DCRPCER_NO_SUCH_SERVICE_GROUP`.

2. If an RPC with domain qualification is used, a transaction cannot be extended even if the process which called the function `dc_rpc_call()` is a transaction. Therefore, to request a service with domain qualification from a transaction, specify `DCRPC_NOTRAN` for `flags` not to extend the transaction. When the local domain is specified for the domain name, the transaction also cannot be extended.
3. When an RPC with domain qualification is used, a service request can be addressed only to a server that receives requests from queue, rather than a server that receives requests from socket.
4. A service request with domain qualification is sent to the domain-alternate schedule service which is activated on the host registered with the `namdomainsetup` command. Obtain the port number of the domain-alternate schedule service from `/etc/services`. If an error occurs while transferring the service request and multiple host names are registered with the `namdomainsetup` command, transfer of the service request is attempted to other hosts sequentially. Even if the RPC with domain qualification terminates normally, an error may occur during transfer to the domain-alternate schedule service.

- Preparation for requesting a service with domain qualification

  Perform the following environment setup for an RPC with domain qualification:

  1. Register the name of the host on which the domain alternate schedule service is activated in the DNS domain data file by using the `namdomainsetup` command.
  2. Define the port number of the domain alternate schedule service in `/etc/services` of the host on which the OpenTP1 which requests a service with domain qualification is activated as follows:

```
OpenTP1scd port-number/tcp
```

  3. Specify the well-known port of the domain alternate schedule service for the `scd_port` operand in the schedule service definition for the OpenTP1 which activates the domain-alternate schedule service.

Note

Assume that you want to perform a transactional RPC on an OpenTP1 system other than the domain specified in the `all_node` clause of the system common definition. In this case, you must ensure that the node identifiers (`node_id` clause of the system common definition) of all OpenTP1 systems in the local domain and remote domain are unique. In addition, all the OpenTP1 systems must be version 03-02 or later. If these conditions are not met, the transaction may not recover properly.

dc_rpc_call_to - Invoke a remote service with a communication destination specified

Format

- ANSI C, C++

```
#include <dcrpc.h>
int dc_rpc_call_to(struct DCRPC_BINDING_TBL *direction,
                   char *group, char *service,char *in,
                   DCULONG *in_len, char *out,
                   DCULONG *out_len, DCLONG flags)
```

- K&R C

```
#include <dcrpc.h>
int dc_rpc_call_to(*direction, *group, *service, *in,
                   *in_len, *out, *out_len, flags)
struct DCRPC_BINDING_TBL  *direction;
char      *group;
char      *service;
char      *in;
DCULONG   *in_len;
char      *out;
DCULONG   *out_len;
DCLONG    flags;
```

Description

The function `dc_rpc_call_to()` requests an SPP service. Like the function `dc_rpc_call()`, this function sets a *service group name* and *service name* as arguments. In addition, it sets the `DCRPC_BINDING_TBL` structure in which a host name or node identifier is specified as an argument. The host name or node identifier specified in the `DCRPC_BINDING_TBL` structure is used as a search key that designates the requested service. This function requests a service from the service function that matches the setting.

However, you cannot add a domain qualification when requesting a service. In all other respects, this function is the same as the function `dc_rpc_call()`.

TP1/Extension 1 must be installed before you can use this facility. Note that operation will be unpredictable if you run the facility while TP1/Extension 1 is not installed.

Arguments whose values are set in the UAP

- `direction`

  Specify the address of the `DCRPC_BINDING_TBL` structure that is to store the search key that designates the requested service. The search key is either a host name or node identifier.

The following shows the format of the DCRPC_BINDING_TBL structure.

```
struct DCRPC_BINDING_TBL {
    char *nid;            /*Storage address for node identifier*/
    char *hostnm;         /*Storage address for host name*/
    short portno;         /*Port number*/
    short filler1;        /*Spare status*/
    DCLONG flags;         /*Attribute*/
    DCLONG filler2[4];    /*Spare status*/
};
```

- nid

  Specify the address of the area that stores the node identifier of the requested service node when you want to set a node identifier as the search key. End the character string with a null character. The null character is not counted in the length of the character string.

  The node identifier must be the name specified for node_id in the system common definition. The host name of the requested service node must exist in the global domain# (a collection of node names specified for the all_node operand of the system common definition).

  When you do not intend to set a node identifier as the search key, specify address 0 for nid.

- hostnm

  Specify the address of the area that stores the host name of the requested service node when you want to set a host name as the search key. You can specify a character string containing between 1 and 255 characters as the host name. This character string can consist of alphanumeric characters and special symbols, the period, and the hyphen (except in the IP address format). End the character string with a null character. The null character is not counted in the length of the character string. The name of the specified host is one that can be mapped to an IP address with the /etc/hosts file or DNS.

  It is optional whether the host name of the requested service node is specified in the global domain# (a collection of node names specified for the all_node operand of the system common definition).

  When you do not intend to set a host name as the search key, specify address 0 for hostnm.

- portno

  Specify the port number (the value specified for name_port in the system common definition) of the name service of the requested service node when you want to set a host name as the search key. The value specified for portno is valid only when DCRPC_NAMPORT is specified for flags in the DCRPC_BINDING_TBL

structure. If you specify 0 for portno or specify DCNOFLAGS for flags in the DCRPC_BINDING_TBL structure, the port number of the name service at the request source and the port number of the name service at the requested service must match.

When you set a node identifier as the search key, the value specified for portno is ignored.

- flags

  Specify DCNOFLAGS.

  If you specified a value for portno, specify DCRPC_NAMPORT.

  The areas filler1 and filler2 were created to allow expandability, so you need not set values for these areas. (Do not use the member names filler1 and filler2.)

#

This global domain means a group of the following node names.

When N is specified in the name_domain_file_use operand in the system common definition:

The global domain is a group of node names specified in the all_node and all_node_ex operands in the system common definition.

When Y is specified in the name_domain_file_use operand in the system common definition:

The global domain is a group of node names specified in the domain definition files. The domain definition files are stored under the following directories:

- Domain definition file for all_node

  $DCCONFPATH/*dcnamnd-directory*

- Domain definition file for all_node_ex

  $DCCONFPATH/*dcnamndex-directory*

You can create the DCRPC_BINDING_TBL structure to be specified for direction in the function dc_rpc_call_to() by using the DCRPC_BINDTBL_SET function or DCRPC_DIRECT_SCHEDULE function. For details, see DCRPC_BINDTBL_SET and DCRPC_DIRECT_SCHEDULE.

■ group

Specify the SPP service group name with an ASCII character string of upto 31 bytes. End the character string with a null character. The null character is not counted in the length of the character string.

- service

  Specify the SPP service name with an ASCII character string of up to 31 bytes. End the character string with a null character. The null character is not counted in the length of the character string.

- in

  Specify the input parameter of the service.

- in_len

  Specify the input parameter length of the service within the range from 1 to DCRPC_MAX_MESSAGE_SIZE#. DCRPC_MAX_MESSAGE_SIZE is defined in dcrpc.h.

  #: If you used the rpc_max_message_size operand, the value of this data area is the value specified in the rpc_max_message_size operand and not the value of DCRPC_MAX_MESSAGE_SIZE (1 megabyte).

- out

  Specify the area for the response from the service function. This area will receive the response from the service function.

- out_len

  Specify the length of the response from the service within the range from 1 to DCRPC_MAX_MESSAGE_SIZE#. DCRPC_MAX_MESSAGE_SIZE is defined in dcrpc.h.

  #: If you used the rpc_max_message_size operand, the value of this data area is the value specified in the rpc_max_message_size operand and not the value of DCRPC_MAX_MESSAGE_SIZE (1 megabyte).

  Even if the RPC is the non-response-type, you must specify the address of the area for which the length of the response from the service is specified. Note that the length of the response from the service must be 0.

- flags

  Specify the RPC mode and option in the following format:

```
{DCNOFLAGS|DCRPC_NOWAIT|DCRPC_NOREPLY|DCRPC_CHAINED}
[|DCRPC_TPNOTRAN]
```

DCNOFLAGS

Synchronous response-type RPC

DCRPC_NOWAIT

Asynchronous response-type RPC

DCRPC_NOREPLY

Non-response-type RPC

DCRPC_CHAINED

Chained RPC

DCRPC_TPNOTRAN

Specify this option to prevent processing requested from a transaction by a service request from being handled as a transaction. Alternatively, specify this option when you want to use the DCRPC_DIRECT_SCHEDULE function to create the DCRPC_BINDING_TBL structure, and to request a service from a user server with the non-transaction attribute. Here, a user server has the non-transaction attribute when N is specified for atomic_update in the user service definition or Y is set for jnl_fileless_option in the system common definition.

This value must be ORed with the type of RPC.

Example:

When a nontransaction service is requested by using a synchronous response-type RPC, specify flags as follows:

```
DCNOFLAGS|DCRPC_TPNOTRAN
```

Arguments whose values are returned from server UAP

- out

  The response set by the service function is returned.

- out_len

  The length of the response set by the service function is returned.

Return values

See the return values for the function dc_rpc_call().

The return values for the function dc_rpc_call_to() include the following causes in addition to those given in the return values for the function dc_rpc_call().

Return value	Return value (numeric)	Explanation
DCRPCER_INVALID_ARGS	-301	The value specified for an argument is invalid.
		The host name specified in hostnm of the DCRPC_BINDING_TBL structure cannot be mapped to an IP address with the /etc/hosts file or DNS.

Return value	Return value (numeric)	Explanation
		The DCRPC_BINDING_TBL structure specified for the first argument of the function dc_rpc_call_to() was created using the DCRPC_DIRECT_SCHEDULE function and 0 was specified for hostnm in the DCRPC_DIRECT_SCHEDULE function.
DCRPCER_NO_SUCH_SERVICE_GROUP	-310	The service group specified in group is not defined. Or, the dc_rpc_call_to() function was executed using a facility that is not supported by the service group specified in group.
		The node identifier specified for nid in the DCRPC_BINDING_TBL structure does not exist in the global domain# (a collection of node names specified for the all_node operand of the system common definition).
DCRPCER_TRNCHK_EXTEND	-372	The transaction branch cannot be started since it exceeds the maximum number of transaction branches that can be activated concurrently.
		The transaction branch cannot be started since it exceeds the maximum number of child transaction branches that can be activated from one transaction branch.
		DCRPC_TPNOTRAN is not specified for flags when a service with domain qualification specified in a transaction is requested.
		Transaction branching cannot start because the resource manager (RM) has encountered an error.
		The function DCRPC_DIRECT_SCHEDULE was used to create the DCRPC_BINDING_TBL structure, and a service was requested from a user server with the non-transaction attribute (atomic_update is N in the user service definition or jnl_fileless_option is Y in the system common definition). However, a disjunction with DCRPC_TPNOTRAN was not specified for the flags argument of the function dc_rpc_call_to.
		In the System Environment window of TP1/LiNK, the **Transaction Facility** item is not set to **Yes**.

#

This global domain means a group of the following node names.

When N is specified in the name_domain_file_use operand in the system common definition:

The global domain is a group of node names specified in the all_node and all_node_ex operands in the system common definition.

When Y is specified in the name_domain_file_use operand in the system common definition:

The global domain is a group of node names specified in the domain definition files. The domain definition files are stored under the following directories:

- Domain definition file for all_node

  $DCCONFPATH/*dcnamnd-directory*

- Domain definition file for all_node_ex

  $DCCONFPATH/*dcnamndex-directory*

Other related items

See the items for the function dc_rpc_call().

Notes

1. Take care when specifying a value for hostnm in the DCRPC_BINDING_TBL structure, hostnm in the DCRPC_BINDING_SET function, or hostnm in the DCRPC_DIRECT_SCHEDULE function under a multi-homed host mode in which multiple LAN adaptors are connected within a single machine. In such a case, do not specify any host name on the local machine other than the host name specified for my_host in the system common definition. If you specify any other host name, _operation will be unpredictable.
2. If you specify both a host name and node identifier in the DCRPC_BINDING_TBL structure, the host name is valid and the node identifier is ignored.
3. If you specify 0 for both the host name and node identifier in the DCRPC_BINDING_TBL structure, operation is exactly the same as for the function dc_rpc_call().
4. To request a service directly from a user server managed by the schedule service, be sure to create the DCRPC_BINDING_TBL structure using the DCRPC_DIRECT_SCHEDULE function.
5. If you create the DCRPC_BINDING_TBL structure using the DCRPC_DIRECT_SCHEDULE function and request a service from a user server that receives requests from socket (socket is specified for receive_from in the user service definition), the function dc_rpc_call_to() returns with the error DCRPCER_SERVICE_NOT_UP.

6. This note applies when you call the function dc_rpc_call_to() with the DCRPC_BINDING_TBL structure created by the function DCRPC_DIRECT_SCHEDULE specified in order to request a service from a user server with the non-transaction attribute. Here, a user server has the non-transaction attribute when N is specified for the atomic_update operand in the user service definition or Y is specified for the jnl_fileless_option operand in the system common definition. In this case, you must specify a disjunction with DCRPC_TPNOTRAN in the flags argument of the function dc_rpc_call_to(). Failure to specify disjunction causes the function dc_rpc_call_to() to return the error DCRPCER_TRNCHK_EXTEND.

7. If you call the function dc_rpc_call_to() in which you specified a DCRPC_BINDING_TBL structure created using the DCRPC_DIRECT_SCHEDULE function, OpenTP1 running the requested service must be Version 03-02 or later. Operation is not guaranteed if the version is earlier than 03-02.

8. You cannot issue an RPC that has a domain qualification. Specifying DCRPC_DOMAIN for flags in the function dc_rpc_call_to() causes the function to return the error DCRPCER_INVALID_ARGS.

9. In the following case, the function dc_rpc_call_to() may return the error DCRPCER_TIMED_OUT: You used a host name as the search key when calling the function dc_rpc_call_to() from a service group on a node that is not specified in the all_node operand of the system common definition, and subsequently you stopped or restarted OpenTP1 running on the called node and again called the function dc_rpc_call_to() from the same service group using a host name as the search key.

10. When the function dc_rpc_call_to() is requested by specifying the DCRPC_BINDING_TBL structure that was created with the DCRPC_DIRECT_SCHEDULE function for direction of the function dc_rpc_call_to(), the rpc_retry operand becomes invalid.

11. The performance verification trace can be obtained when the function dc_rpc_call_to() is requested by specifying the DCRPC_BINDING_TBL structure that was created with the DCRPC_DIRECT_SCHEDULE function for direction of the function dc_rpc_call_to(), but it cannot be linked to the information about the UAP performance verification trace in the request destination. The serial number of the performance verification trace obtained with the client UAP is not inherited in the server UAP.

DCRPC_BINDTBL_SET and DCRPC_DIRECT_SCHEDULE - Create the DCRPC_BINDING_TBL structure

Format

- ANSI C, C++

```
#include <dcrpc.h>
void DCRPC_BINDTBL_SET(struct DCRPC_BINDING_TBL *direction,
                       char *nid, char *hostnm,
                       short portno, DCLONG flags)
void DCRPC_DIRECT_SCHEDULE(struct DCRPC_BINDING_TBL
                           *direction, char *hostnm,
                           short scdport, DCLONG flags)
```

- K&R C

```
#include <dcrpc.h>}
void DCRPC_BINDTBL_SET(*direction, *nid, *hostnm,
                       portno, flags)
struct DCRPC_BINDING_TBL *direction;
char      *nid;
char      *hostnm;
short     portno;
DCLONG    flags;
void DCRPC_DIRECT_SCHEDULE(*direction, *hostnm, scdport,
                           flags)
struct DCRPC_BINDING_TBL  *direction;
char      *hostnm;
short     scdport;
DCLONG    flags;
```

Description

Create the DCRPC_BINDING_TBL structure to be specified for the first argument of the function dc_rpc_call_to() by using one of the following functions:

- DCRPC_BINDTBL_SET function

  Specify the node identifier (nid) or host name (hostnm) of the requested service node in the DCRPC_BINDING_TBL structure to create the first argument for the function dc_rpc_call_to().

- DCRPC_DIRECT_SCHEDULE function

  Specify the host name (hostnm) of the requested service node and the port number (scdport) of the specified schedule service in the DCRPC_BINDING_TBL structure to create the first argument for the function dc_rpc_call_to().

When you call the function dc_rpc_call_to() in which you specified a

DCRPC_BINDING_TBL structure created using the DCRPC_DIRECT_SCHEDULE function, OpenTP1 sends a service request directly to the user server managed by the specified schedule service. However, you can use a DCRPC_BINDING_TBL structure creating using the DCRPC_DIRECT_SCHEDULE function only when requesting a service from a queue-receiving (queue is specified for receive_from in the user service definition) user server.

You must observe numerous rules when calling the function dc_rpc_call_to() in which you specified a DCRPC_BINDING_TBL structure created using the DCRPC_DIRECT_SCHEDULE function. For example, you must be aware of the version of OpenTP1 running the requested service and the transaction attribute of the user server. For details, see the notes for the function dc_rpc_call_to().

Arguments whose values are set in the UAP

- direction

  Specify the address of the DCRPC_BINDING_TBL structure used for the first argument of the function dc_rpc_call_to().

- nid

  In the DCRPC_BINDTBL_SET function, specify the address of the area that stores the node identifier when you want to set a node identifier as the search key that designates the requested service. End the character string with a null character. The null character is not counted in the length of the character string.

  The node identifier must be the name specified for node_id in the system common definition and the host name of the requested service node must exist in the global domain# (a collection of node names specified for the all_node operand of the system common definition).

  When you do not intend to set a node identifier as the search key, specify address 0 for nid.

  #

  This global domain means a group of the following node names.

  When N is specified in the name_domain_file_use operand in the system common definition:

  The global domain is a group of node names specified in the all_node and all_node_ex operands in the system common definition.

  When Y is specified in the name_domain_file_use operand in the system common definition:

  The global domain is a group of node names specified in the domain definition files. The domain definition files are stored under the following directories:

- Domain definition file for `all_node`

  `$DCCONFPATH/`*dcnamnd-directory*

- Domain definition file for `all_node_ex`

  `$DCCONFPATH/`*dcnamndex-directory*

■ `hostnm`

Specify the address of the area that stores the host name of the requested service node. You can specify a character string containing between 1 and 255 characters as the host name. End the character string with a null character. The null character is not counted in the length of the character string. The name of the specified host is one that can be mapped to an IP address with the `/etc/hosts` file or DNS.

It is optional whether the host name of the requested service node is specified in the global domain# (a collection of node names specified for the `all_node` operand of the system common definition).

When you do not intend to set a host name as the search key that designates the requested service in the `DCRPC_BINDTBL_SET` function, specify address 0 for `hostnm`.

Be sure to specify `hostnm` in the `DCRPC_DIRECT_SCHEDULE` function. If you specify address 0 for `hostnm` in the `DCRPC_DIRECT_SCHEDULE` function, calling the function `dc_rpc_call_to()` with the `DCRPC_BINDING_TBL` structure specified causes the function to return the error `DCRPCER_INVALID_ARGS`.

#

This global domain means a group of the following node names.

When `N` is specified in the `name_domain_file_use` operand in the system common definition:

The global domain is a group of node names specified in the `all_node` and `all_node_ex` operands in the system common definition.

When `Y` is specified in the `name_domain_file_use` operand in the system common definition:

The global domain is a group of node names specified in the domain definition files. The domain definition files are stored under the following directories:

- Domain definition file for `all_node`

  `$DCCONFPATH/`*dcnamnd-directory*

- Domain definition file for `all_node_ex`

  `$DCCONFPATH/`*dcnamndex-directory*

- portno
  - When you set a host name as the search key in the DCRPC_BINDTBL_SET function

    Specify the port number (the value specified for name_port operand in the system common definition) of the name service of the OpenTP1 system running the requested service. If the port number of the name service at the requested service matches the port number of the name service at the request source, specify 0.
  - When you set a node identifier as the search key in the DCRPC_BINDTBL_SET function

    Specify 0 for portno. If you omit the port number (the value specified for the all_node operand in the system common definition) at the requested service, the port number (the value specified for the name_port operand in the system common definition) of the name service at the requested service and the port number of the name service at the request source must match.
- scdport

  For scdport in the DCRPC_DIRECT_SCHEDULE function, specify the port number of the schedule service provided by the OpenTP1 system that offers the requested service (the value assigned to scd_port in the schedule service definition for the requested service). If you specify 0, the transmission destination port number is assumed by default to be the value assigned to scd_port specified in the schedule service definition on the service request issuer. Therefore, before you can specify 0 for scdport in the DCRPC_DIRECT_SCHEDULE function, the OpenTP1 system of the service request issuer must be active and scd_port must be specified in the schedule service definition for the OpenTP1 system.
- flags

  Specify DCNOFLAGS.

Other related items

See the items for the function dc_rpc_call_to().

Notes

1. The DCRPC_BINDTBL_SET function and DCRPC_DIRECT_SCHEDULE function are provided for setting the DCRPC_BINDING_TBL structure to be specified for the first argument of the function dc_rpc_call_to().
2. Details of how to check the values specified for the arguments of the DCRPC_BINDTBL_SET function and DCRPC_DIRECT_SCHEDULE function and how to specify the values are given in the description of calling the function dc_rpc_call_to() with the DCRPC_BINDING_TBL structure specified. For

details about the `dc_rpc_call_to()` function, see *dc_rpc_call_to* in *2. Remote procedure call (dc_rpc_~)*.

3. The `DCRPC_BINDTBL_SET` function and `DCRPC_DIRECT_SCHEDULE` function do not acquire any UAP trace.

dc_rpc_close - Terminate an application program

Format

- ANSI C, C++

```
#include <dcrpc.h>
void dc_rpc_close (DCLONG flags)
```

- K&R C

```
#include <dcrpc.h>
void dc_rpc_close (flags)
DCLONG  flags;
```

Description

The function `dc_rpc_close()` closes the environment for using various types of OpenTP1 functions. OpenTP1 functions cannot be used after the function `dc_rpc_close()`.

The function `dc_rpc_close()` must be called in the main function. Call the function `dc_rpc_close()` only once in the process.

The function `dc_rpc_close()` also informs OpenTP1 of normal termination. If a UAP terminates without the function `dc_rpc_close()` called, OpenTP1 assumes that the UAP terminated abnormally. Consequently, the service group might be shut down or the process might be restarted. To make matters worse, various OpenTP1 resources might not be released, which affects the subsequent processing.

If the function `dc_rpc_open()` is called from any UAP used with OpenTP1, the function `dc_rpc_close()` must be called before the UAP terminates with `exit()`.

Call the function `dc_rpc_close()` even if the function `dc_rpc_open()` returns with an error.

After the function `dc_rpc_close()` is called, the function `dc_rpc_open()` cannot be called from the same UAP.

Argument whose value is set in the UAP

- `flags`

  Specify `DCNOFLAGS`.

Return value

There is no return value of the function `dc_rpc_close()`.

dc_rpc_cltsend - Report data to CUP unidirectionally

Format

- ANSI C, C++

```
#include <dcrpc.h>
int  dc_rpc_cltsend (char *hostname, unsigned short port,
                     char *msg,  DCLONG len, DCLONG flags)
```

- K&R C

```
#include <dcrpc.h>
int  dc_rpc_cltsend (hostname, port, msg, len, flags)
char       *hostname;
unsigned  short  port;
char       *msg;
DCLONG     len;
DCLONG     flags;
```

Description

The function `dc_rpc_cltsend()` sends data to the CUP unidirectionally. This function sends data specified for `msg` of the length specified for `len` to the process (CUP) corresponding to the port number of the host specified for `hostname` and `port`. The possible sending data length is in the range of bytes from 0 to `DCRPC_MAX_MESSAGE_SIZE`#.

#: If you used the `rpc_max_message_size` operand, the value of this data area is the value specified in the `rpc_max_message_size` operand and not the value of `DCRPC_MAX_MESSAGE_SIZE` (1 megabyte).

Data sent by the function `dc_rpc_cltsend()` is received by the TP1/Client library function `dc_clt_chained_accept_notification()` or `dc_clt_accept_notification()`. For the function `dc_clt_chained_accept_notification()` or `dc_clt_accept_notification()`, see the manual *OpenTP1 TP1/Client/W, TP1/Client/P*.

Arguments whose values are set in the UAP

- `hostname`

  Specify the name of the host to which data is sent. You can specify a character string containing between 1 and 255 characters as the host name. End the character string with a null character.

- port

  Specify the number of the port to which data is sent.

- msg

  Specify data to be sent.

- len

  Specify the length of data to be sent.

- flags

  Specify DCNOFLAGS.

Return values

Return value	Return value (numeric)	Explanation
DC_OK	0	Normal termination.
DCRPCER_INVALID_ARGS	-301	The value specified for the argument is invalid.
DCRPCER_NET_DOWN	-306	A network error occurred.
DCRPCER_PROTO	-302	The function dc_rpc_open() was not called.
DCRPCER_NO_BUFS	-304	The memory became insufficient.
DCRPCER_MESSAGE_TOO_BIG	-308	The length of data to be sent exceeds DCRPC_MAX_MESSAGE_SIZE#.
DCRPCER_SERVICE_NOT_UP	-314	There is no process at the destination.
		A network error occurred.

#: If you used the rpc_max_message_size operand, the value of this data area is the value specified in the rpc_max_message_size operand and not the value of DCRPC_MAX_MESSAGE_SIZE (1 megabyte).

Notes

1. Use the function dc_rpc_cltsend() only when the process of the destination calls the TP1/Client function dc_clt_chained_accept_notification() or dc_clt_accept_notification() obviously. If the process of the destination is not active, the function dc_rpc_cltsend() returns with an error, giving the return value DCRPCER_SERVICE_NOT_UP.
2. Normal return of the function dc_rpc_cltsend() indicates that sending at RPC communication protocol (TCP/IP) level is completed. Therefore, normal termination of the function dc_rpc_cltsend() does not guarantee that the data

is received normally by the CUP using the function `dc_clt_chained_accept_notification()` or `dc_clt_accept_notification()`.

3. The function `dc_rpc_cltsend()` can report data only to the function `dc_clt_chained_accept_notification()` or `dc_clt_accept_notification()` used by the CUP. Data cannot be sent to SPP processes and local processes.

dc_rpc_discard_further_replies - Reject the receiving of processing results

Format

- ANSI C, C++

```
#include <dcrpc.h>
void  dc_rpc_discard_further_replies (DCLONG flags)
```

- K&R C

```
#include <dcrpc.h>
void  dc_rpc_discard_further_replies (flags)
DCLONG    flags;
```

Description

The function `dc_rpc_discard_further_replies()` specifies that no more responses (which have not been returned) will be received through an asynchronous-response-type RPC (`DCRPC_NOWAIT` specified for `flags` of the function `dc_rpc_call()`). After the function `dc_rpc_discard_further_replies()` is called, returned responses are discarded instead of being received.

To stop receiving further processing results of an asynchronous-response-type RPC, call the function `dc_rpc_discard_further_replies()`. Otherwise, the function `dc_rpc_poll_any_replies()` might receive unnecessary responses.

Use the function `dc_rpc_discard_further_replies()` in the following cases:

- After a response wait timeout occurs, the buffer for shutting down the processing results is released.
- An asynchronous-response-type RPC has been called more than once, but only the first response is necessary.

Argument whose value is set in the UAP

- `flags`

  Specify `DCNOFLAGS`.

Return value

There is no return value of the function `dc_rpc_discard_further_replies()`.

dc_rpc_discard_specific_reply - Reject acceptance of particular processing results

Format

- ANSI C, C++

```
#include <dcrpc.h>
int  dc_rpc_discard_specific_reply (int des, DCLONG flags)
```

- K&R C

```
#include <dcrpc.h>
int  dc_rpc_discard_specific_reply (des, flags)
int        des;
DCLONG     flags;
```

Description

The function dc_rpc_discard_specific_reply indicates that the UAP will no longer receive a specific response which can be returned by an asynchronous-response type RPC (when DCRPC_NOWAIT was specified in flags in the function dc_rpc_call) but has not yet been returned. To specify the asynchronous response whose reception is to be rejected, specify the descriptor returned when an asynchronous-response type RPC returned in des. Of the responses that return after this function is called, responses having the same descriptor as the specified descriptor are discarded without being received.

Arguments whose values are set in the UAP

- des

  Specify the descriptor returned when the function dc_rpc_call (with DCRPC_NOWAIT specified in flags) of an asynchronous-response type RPC terminated normally.

- flags

  Specify DCNOFLAGS.

Return values

Return value	Return value (numeric)	Explanation
DC_OK	0	Normal termination.
DCRPCER_INVALID_ARGS	-301	The value specified for the argument is invalid.

Return value	Return value (numeric)	Explanation
DCRPCER_PROTO	-302	The function dc_rpc_open was not called.
DCRPCER_INVALID_DES	-322	The descriptor specified for des does not exist. An asynchronous-response type RPC corresponding to the specified descriptor was not sent, or a response has already been received through an asynchronous-response type RPC, or reception of a response was rejected.

dc_rpc_get_callers_address - Acquire the node address of a client UAP

Format

- ANSI C, C++

```
#include <dcrpc.h>
void  dc_rpc_get_callers_address (DCULONG *node,
                                  DCLONG flags)
```

- K&R C

```
#include <dcrpc.h>
void  dc_rpc_get_callers_address (node, flags)
DCULONG     *node;
DCLONG      flags;
```

Description

The function `dc_rpc_get_callers_address()` allows the server UAP to acquire the address of the node at which the client UAP process is working. Security checking for the client UAP can be performed using the address obtained by the function `dc_rpc_get_callers_address()`.

The address obtained by the function `dc_rpc_get_callers_address()` cannot be used for sending a service response or error response.

The function `dc_rpc_get_callers_address()` must be called from a service function. Otherwise, processing is unpredictable.

Arguments whose value is set in the UAP

- `flags`

  Specify `DCNOFLAGS`.

Arguments whose value is returned from OpenTP1

- `node`

  The node address of the client UAP is returned.

Return values

There is no return value of the function `dc_rpc_get_callers_address()`.

Note

When both the following conditions are true, the node address of the client UAP returned by the `dc_rpc_get_callers_address()` function might differ from the

node address actually used by the client UAP during communication:

- A service request was accepted using the remote API facility.
- The host containing the client UAP is a multi-homed host mode.

dc_rpc_get_error_descriptor - Acquire the descriptor of an asynchronous response-type RPC request which has encountered an error

Format

■ ANSI C, C++

```
#include <dcrpc.h>
int dc_rpc_get_error_descriptor(DCLONG flags)
```

■ K&R C

```
#include <dcrpc.h>
int dc_rpc_get_error_descriptor (flags)
DCLONG     flags
```

Description

The function `dc_rpc_get_error_descriptor()` acquires the descriptor of an asynchronous response-type RPC request which has encountered an error when it is called just after the function `dc_rpc_poll_any_replies()` without a particular asynchronous response specified returns with an error.

It can acquire the descriptor only when the error has occurred on the SPP.

If an error has occurred on the `dc_rpc_poll_any_replies()` caller, the function `dc_rpc_get_error_descriptor()` cannot acquire the descriptor.

Arguments whose value is set in the UAP

■ `flags`

Specify `DCNOFLAGS`.

Return values

Return value	Return value (numeric)	Explanation
Positive integer		The function acquired the descriptor of the asynchronous response-type RPC request which encountered the error returned by the function `dc_rpc_poll_any_replies()`.
`0`		The function failed to acquire the descriptor of the asynchronous response-type RPC request which encountered the error returned by the function `dc_rpc_poll_any_replies()`.

Return value	Return value (numeric)	Explanation
DCRPCER_INVALID_ARGS	-301	The value specified for the argument is invalid.

dc_rpc_get_gateway_address - Acquire the node address of a gateway

Format

- ANSI C, C++

```
#include <dcrpc.h>
int dc_rpc_get_gateway_address(DCULONG *node,DCLONG flags)
```

- K&R C

```
#include <dcrpc.h>
int dc_rpc_get_gateway_address(node, flags)
DCULONG    *node;
DCLONG     flags;
```

Description

The function `dc_rpc_get_gateway_address` acquires the node address of a gateway from the server UAP when a service request was received from a client UAP via a gateway, such as the application gateway FireWall.

The server UAP can acquire the node address of the gateway when a service was requested using the remote API facility.

A service response or error response cannot be sent using the address that is returned for the function `dc_rpc_get_gateway_address`.

Call the function `dc_rpc_get_gateway_address` from the service function. Processing is not guaranteed if the function is called from a function other than the service function.

Arguments whose values are set in the UAP

- `node`

  Specify the address of the area to which the node address of the gateway is to be returned.

- `flags`

  Specify `DCNOFLAGS`.

Arguments whose values are returned from OpenTP1

- `node`

  The node address of the gateway is returned. The value 0 is set when the remote API facility was not used.

Return values

Return value	Return value (numeric)	Explanation
DC_OK	0	Normal termination
DCRPCER_INVLAID_ARGS	-301	The value specified for the argument is invalid.
DCRPCER_PROTO	-302	The function dc_rpc_get_gateway_address was not called from the service function.

dc_rpc_get_service_prio - Reference the schedule priority of a service request

Format

■ ANSI C, C++

```
#include <dcrpc.h>
int  dc_rpc_get_service_prio (void)
```

■ K&R C

```
#include <dcrpc.h>
int  dc_rpc_get_service_prio()
```

Description

The function `dc_rpc_get_service_prio()` references that schedule priority of a service request which was set by the function `dc_rpc_set_service_prio()`. The value obtained by this function remains unchanged until the UAP calls the function `dc_rpc_set_service_prio()` again.

The function `dc_rpc_get_service_prio()` returns the default value (4) in the following cases:

- The UAP has not called the function `dc_rpc_set_service_prio()`.
- The function `dc_rpc_set_service_prio()` has been called with 0 specified for the argument `prio`.

Return values

Return value	Explanation
Positive integer	Schedule priority set by the function `dc_rpc_set_service_prio()`, in the range from 1 to 8.
Other than the above	An unprecedented error (e.g., program damage) occurred.

dc_rpc_get_watch_time - Reference the service response waiting interval

Format

■ ANSI C, C++

```
#include <dcrpc.h>
int  dc_rpc_get_watch_time (void)
```

■ K&R C

```
#include <dcrpc.h>
int  dc_rpc_get_watch_time()
```

Description

The function `dc_rpc_get_watch_time()` references the current response waiting interval of a service request. This function is used for saving the current value of the service response waiting interval before temporarily changing it using the function `dc_rpc_set_watch_time()`.

The function returns the service response waiting interval changed by the function `dc_rpc_set_watch_time()`. When the interval has not been changed, the following value is returned:

- For TP1/Server Base: Value of the `watch_time` operand in the system common definition
- For TP1/LiNK: 180 seconds

The value obtained by this function can be used by the OpenTP1 function `dc_rpc_call()`.

Return values

Return value	Return value (numeric)	Explanation
`DC_OK`	`0`	The service response waiting interval means indefinite wait.
Positive integer		Current the service response waiting interval.
Other than the above		An unprecedented error (e.g., program damage) occurred.

dc_rpc_mainloop - Start an SPP service

Format

■ ANSI C, C++

```
#include <dcrpc.h>
int  dc_rpc_mainloop (DCLONG flags)
```

■ K&R C

```
#include <dcrpc.h>
int  dc_rpc_mainloop(flags)
DCLONG     flags;
```

Description

The function `dc_rpc_mainloop()` starts the receiving of service requests to a service function of the SPP which is being executed in the process. The function `dc_rpc_mainloop()` must be called in the main function. Call the function `dc_rpc_mainloop()` only once in the process.

The function `dc_rpc_mainloop()` does not return until it receives a termination request from OpenTP1. The function `dc_rpc_mainloop()` receives a termination request from OpenTP1 in the following cases:

- Termination processing starts because one of the following OpenTP1 stop commands has been accepted:

  `dcstop` command (normal termination)

  `dcstop -n` command (forced normal termination)

  `dcstop -a` command (planned termination A)

  `dcstop -b` command (planned termination B)

- The following server stop command is entered to start termination processing for the processes of the SPP that called the function `dc_rpc_mainloop()`:

  `dcsvstop` command (normal termination)

- OpenTP1 terminates the processes of the SPP that called the function `dc_rpc_mainloop()` because the number of processes exceeds the maximum number specified in the user service definition.
- Service processing terminates if the SPP is executing under a nonresident process.
- If loads on SPPs are distributed in a multiserver configuration, service requests addressed to the present service group are reduced.

Argument whose value is set in the UAP

- flags

  Specify DCNOFLAGS.

Return values

Return value	Return value (numeric)	Explanation
DC_OK	0	A termination request was received from OpenTP1. Execute termination processing for the SPP immediately, then call the function dc_rpc_close() and exit().
DCRPCER_INVALID_ARGS	-301	The value specified for the argument is invalid.
DCRPCER_PROTO	-302	The function dc_rpc_open() was not called.
		The function dc_rpc_mainloop() or the function dc_mcf_mainloop() was called.
DCRPCER_FATAL	-303	The SPP service could not be started.

Notes

The function dc_rpc_mainloop() returns when it receives a termination request from OpenTP1. However, the function dc_rpc_mainloop() does not return but the process terminates in the following cases:

1. The SPP enters a termination process because the OpenTP1 forced termination command (dcstop -f command) or server forced termination command (dcsvstop -f command) is executed.
2. A process terminates abnormally because the UAP or OpenTP1 malfunctions.
3. The service function issues abort() or exit().
4. Hardware, the operating system, or OpenTP1 causes an error.

Even if the SPP is created in such a way that termination processing will be executed after the function dc_rpc_mainloop() terminates normally, the processing is not executed in the above cases.

dc_rpc_open - Start an application program

Format

- ANSI C, C++

```
#include <dcrpc.h>
int  dc_rpc_open (DCLONG flags)
```

- K&R C

```
#include <dcrpc.h>
int  dc_rpc_open (flags)
DCLONGflags;
```

Description

The function `dc_rpc_open()` prepares to use the various types of OpenTP1 functions. The function `dc_rpc_open()` must be called in the main function. Call the function `dc_rpc_open()` only once in the process.

To initialize in the main function:

1. Open the entry point for communication between processes.
2. Acquire shared memory used with OpenTP1.
3. Post the UAP start to OpenTP1 to request OpenTP1 to supervise processes.
4. Initialize the OpenTP1 facilities to be used according to the UAP environment settings.

If the transaction attribute is specified in the user service definition, the OpenTP1 transaction service and the process service must be in progress at the node. The function `dc_rpc_open()` can be called only after OpenTP1 starts normally when the OS starts or after OpenTP1 is started normally by entering the `dcstart` command. If the function `dc_rpc_open()` is called before the normal start of OpenTP1, the function returns with the error value `DCRPCER_OLTF_NOT_UP`. In this case, the function `dc_rpc_call()` cannot be used.

UAP trace is acquired for all OpenTP1 functions called after the function `dc_rpc_open()` terminates normally. If the function `dc_rpc_open()` returns with an error, the UAP trace is not always acquired.

Argument whose value is set in the UAP

- `flags`

  Specify `DCNOFLAGS`.

Return values

Return value	Return value (numeric)	Explanation
DC_OK	0	Normal termination.
DCRPCER_INVALID_ARGS	-301	The value specified for the argument is invalid.
DCRPCER_PROTO	-302	The function dc_rpc_open() was called.
DCRPCER_FATAL	-303	Initialization was unsuccessful. OpenTP1 functions can no longer be used.
DCRPCER_OLTF_NOT_UP	-315	OpenTP1 of the node at which the UAP exists was not executed.
DCRPCER_SEC_INIT	-371	An error occurred in initialization of the security environment of the OpenTP1 that used the security facility.
DCRPCER_STANDBY_END	-369	The end of standby status was requested for a server in the standby system.

Example

```
#include <dcrpc.h>
main(){
if(dc_rpc_open(DCNOFLAGS) <0){
  fputs("cannot begin usrserv1",stderr);
goto RPC_CLOSE;
}
if(dc_rpc_mainloop(DCNOFLAGS)<0)
  fputs("cannot begin usrserv1",stderr);
/*The service function is called and executed.*/
/*In the mean time, control does not return to the main function.*/
RPC_CLOSE:
dc_rpc_close(DCNOFLAGS);
}
```

dc_rpc_poll_any_replies - Receive processing results in asynchronous mode

Format

- ANSI C, C++

```
#include <dcrpc.h>
int  dc_rpc_poll_any_replies (int des, DCLONG timeout,
                              DCLONG flags)
```

- K&R C

```
#include <dcrpc.h>
int  dc_rpc_poll_any_replies (des, timeout, flags)
int        des;
DCLONG     timeout;
DCLONG     flags;
```

Description

The function `dc_rpc_poll_any_replies()` receives the processing results of a service requested through an asynchronous-response-type RPC (`DCRPC_NOWAIT` specified for `flags` of the function `dc_rpc_call()`).

To designate a specific asynchronous response received, specify `DCRPC_SPECIFIC_MSG` for `flags`. If this flag is set, the function `dc_rpc_poll_any_replies()` receives the response of the asynchronous response-type RPC which returned the descriptor specified for `des`.

Not to designate a specific asynchronous response received, specify `DCNOFLAGS` for `flags`. In this case, the value specified for des is ignored. When the function `dc_rpc_poll_any_replies()` with `DCNOFLAGS` specified for `flags` terminates normally, it returns the same value as the descriptor of the received asynchronous response.

The function `dc_rpc_poll_any_replies()` returns in the following cases:

- A response is received from an asynchronous-response RPC.
- A response wait timeout occurs (the response wait time specified in `timeout` has elapsed).

When the function `dc_rpc_poll_any_replies()` terminates normally, the received response is stored in the response area specified in the function `dc_rpc_call()` using the asynchronous response-type RPC.

The following items are described after the list of return values. See each description For details on the function `dc_rpc_poll_any_replies()`.

(1) timeout, an argument of the function dc_rpc_poll_any_replies()

(2) Timing when the function dc_rpc_poll_any_replies() results in error

(3) Specification for returning the value DCRPCER_SERVICE_TERMINATED

(4) Relationship between error return values and synchronization point processing

(5) When a response cannot be received by the function dc_rpc_poll_any_replies()

(6) Notes on using the function dc_rpc_poll_any_replies()

Arguments whose values are set in the UAP

- des

  Specify the descriptor which was normally returned by the function dc_rpc_call() (DCRPC_NOWAIT specified for flags) of asynchronous response-type RPC. If DCNOFLAGS is specified for flags, the value set here will be ignored.

- timeout

  Specify wait time (in seconds or milliseconds) until the results of the function dc_rpc_call() of an asynchronous-response-type RPC are returned. The specified wait time must be in the range from -1 to the maximum value which can be indicated by DCLONG type.

  When the function dc_rpc_poll_any_replies() receives an asynchronous response, the response waiting interval specified in the UAP is not referenced.

  If 0 is specified here, DCNOFLAGS or DCRPC_SPECIFIC_MSG is specified for flags, and no response is returned, then the function dc_rpc_poll_replies() will immediately return with the return value DCRPCER_TIMED_OUT. If DCRPC_WAIT_MILLISEC is specified for flags, the wait time will be 50 milliseconds.

  When -1 is specified, the function dc_rpc_poll_any_replies() continues to wait until a response is returned.

- flags

  Use the following format:

```
{DCNOFLAGS|DCRPC_SPECIFIC_MSG}[|DCRPC_WAIT_MILLISEC]
```

  DCNOFLAGS

  Asynchronous responses received by the function dc_rpc_poll_any_replies() will not be identified.

  DCRPC_SPECIFIC_MSG

The response to the asynchronous-response-type RPC which returned the descriptor specified for des will be received.

DCRPC_WAIT_MILLISEC

The wait time specified by timeout is assumed to be in milliseconds.

Return values

Return value	Return value (numeric)	Explanation
DC_OK	0	Normal termination. This value is returned when the function dc_rpc_poll_any_replies() with DCRPC_SPECIFIC_MSG specified for flags terminates normally.
Positive integer		Indicates the descriptor of the received asynchronous response. This value is returned when the function dc_rpc_poll_any_replies() with DCNOFLAGS specified for flags terminates normally.
DCRPCER_ALL_RECEIVED	-321	The results of processing for the service requested with asynchronous response-type RPCs are received completely.
DCRPCER_INVALID_DES	-322	The descriptor specified for des does not exist. This value is returned when DCRPC_SPECIFIC_MSG is specified for flags.
DCRPCER_INVALID_ARGS	-301	The value specified for the argument is invalid.
DCRPCER_PROTO	-302	The function dc_rpc_open() was not called.
DCRPCER_NO_BUFS	-304	The memory became insufficient.
DCRPCER_NET_DOWN	-306	A network error occurred.
DCRPCER_TIMED_OUT	-307	The function dc_rpc_call() or dc_gwf_call() encountered timeout.
		An SPP to which the service request was addressed terminated abnormally before completion of the requested service.
DCRPCER_MESSAGE_TOO_BIG	-308	The input parameter length specified for in_len of the function dc_rpc_call() or dc_gwf_call() exceeded the maximum.
DCRPCER_REPLY_TOO_BIG	-309	The returned response is longer than the area prepared by the client UAP.

Return value	Return value (numeric)	Explanation
DCRPCER_NO_SUCH_SERVICE_GROUP	-310	The service group specified in group of the dc_rpc_call() or dc_gwf_call() function is not defined. Or, the dc_rpc_call() or dc_gwf_call() function was executed using a facility that is not supported by the service group specified in group.
DCRPCER_NO_SUCH_SERVICE	-311	The service name specified in service is not defined in the SPP that requested the service.
DCRPCER_SERVICE_CLOSED	-312	The service group containing the service of which name is specified for service of the function dc_rpc_call() or dc_gwf_call() is in shutdown state.
DCRPCER_SERVICE_TERMINATING	-313	The service specified for service of the function dc_rpc_call() or dc_gwf_call() is being terminated.
DCRPCER_SERVICE_NOT_UP	-314	The UAP process of the service specified for service of the function dc_rpc_call() or dc_gwf_call() is not active.
		An SPP to which the service request was addressed terminated abnormally before completion of the requested service when -1 is specified for timeout.
DCRPCER_OLTF_NOT_UP	-315	The OpenTP1 at the node containing the service specified for service of the function dc_rpc_call() or dc_gwf_call() is not active. The cause may be one of the following: abnormal termination, being-suspended, being-terminated, or communication error.
DCRPCER_SYSERR_AT_SERVER	-316	A system error occurred in the specified service for the function dc_rpc_call() or dc_gwf_call().
DCRPCER_SYSERR	-318	A system error occurred.
DCRPCER_NO_BUFS_AT_SERVER	-317	The memory became insufficient in the specified service for the function dc_rpc_call() or dc_gwf_call().
DCRPCER_INVALID_REPLY	-319	The length of the response returned from the service function to the OpenTP1 is not in the range from 1 to DCRPC_MAX_MESSAGE_SIZE#.
DCRPCER_OLTF_INITIALIZING	-320	The OpenTP1 at the node to which the service request is addressed is being started.

Return value	Return value (numeric)	Explanation
DCRPCER_NO_BUFS_RB	-323	The memory became insufficient. If this value is returned, the transaction branch cannot be committed.
DCRPCER_SYSERR_RB	-324	A system error occurred. If this value is returned, the transaction branch cannot be committed.
DCRPCER_SYSERR_AT_SERVER_RB	-325	A system error occurred when the specified service was executed. If this value is returned, the transaction branch cannot be committed.
DCRPCER_REPLY_TOO_BIG_RB	-326	The returned response is too large to be stored in the area prepared by the client UAP. If this value is returned, the transaction branch cannot be committed.
DCRPCER_TRNCHK	-327	The transaction attributes of multiple SPPs do not match in an environment where the internode load-balancing facility and the extended internode load-balancing facility are in use. This return value is only returned when the service request is addressed to an SPP that uses the internode load-balancing facility and the extended internode load-balancing facility.
DCRPCER_NO_SUCH_DOMAIN	-328	The domain name of the service group name with domain qualification is invalid.
DCRPCER_NO_PORT	-329	When a service is requested with domain qualification, the port number of the domain-alternate schedule service is not found.
DCRPCER_SERVER_BUSY	-356	The server that receives requests from socket to which the service request is addressed cannot receive the service request.
DCRPCER_TESTMODE	-366	When the online tester was in use, a service request was issued from a UAP in test mode to an SPP in nontest mode or from a UAP in nontest mode to an SPP in test mode.
DCRPCER_SECCHK	-370	An SPP to which the service request is addressed is protected with the security facility. The UAP that requests the service by using the function dc_rpc_call() or dc_gwf_call() has no access permission for the SPP.
DCRPCER_TRNCHK_EXTEND	-372	The transaction branch cannot be started since it exceeds the maximum number of transaction branches which can be activated concurrently.

Return value	Return value (numeric)	Explanation
		The transaction branch cannot be started since it exceeds the maximum number of child transaction branches which can be activated from one transaction branch.
		Transaction branching cannot start because the resource manager (RM) has encountered an error.
DCRPCER_SERVICE_TERMINATED	-378	The SPP from which a service was requested terminated abnormally before processing was completed. This value is returned only for the client UAP having the rpc_extend_function operand specified as 00000001. The operand is in the user service definition. If 00000000 is specified in the rpc_extend_function operand, or the operand is omitted, DCRPCER_TIMED_OUT or DCRPCER_SERVICE_NOT_UP is returned rather than this value.

#: If you used the rpc_max_message_size operand, the value of this data area is the value specified in the rpc_max_message_size operand and not the value of DCRPC_MAX_MESSAGE_SIZE (1 megabyte).

(1) timeout, an argument of the function dc_rpc_poll_any_replies()

The monitoring time for receiving an asynchronous response is reset each time a response is returned. Therefore, when a specific asynchronous response received is designated (DCRPC_SPECIFIC_MSG is specified for flags), a response may be received even if the time specified for timeout has elapsed. Alternatively, the function dc_rpc_poll_any_replies() may not return with an error, giving the return value DCRPCER_TIMED_OUT even if the time specified for timeout has elapsed.

(2) Timing when the function dc_rpc_poll_any_replies() results in error

The following explains the timing when an error is returned from the client UAP if the SPP to which the service request is addressed terminates abnormally.

If an SPP to execute a service terminates abnormally before completion of the processing, the function dc_rpc_poll_any_replies() returns with an error, giving the return value DCRPCER_TIMED_OUT. If -1 is specified for timeout, an argument of the function dc_rpc_poll_any_replies(), the function returns with an error, giving the return value DCRPCER_SERVICE_NOT_UP.

- When the function dc_rpc_poll_any_replies() results in error due to time monitoring for the function

  In the following cases, the function returns with an error, giving the return value DCRPCER_TIMED_OUT, after the time specified for timeout, an argument of the function dc_rpc_poll_any_replies(), has elapsed:

  - The entire OpenTP1 at the node containing the SPP terminates abnormally.
  - An error occurs before the server UAP receives service request data or before the client UAP receives the result after the server UAP processing is completed.

(3) Specification for returning the value DCRPCER_SERVICE_TERMINATED

You may want to determine whether the SPP that requested a service terminated abnormally before completion of processing based on a returned value other than DCRPCER_TIMED_OUT or DCRPCER_SERVICE_NOT_UP. If so, specify 00000001 in the rpc_extend_function operand of the user service definition. This specification returns DCRPCER_SERVICE_TERMINATED if the above error occurs. If 00000000 is specified in the rpc_extend_function operand, or the operand is omitted, DCRPCER_TIMED_OUT or DCRPCER_SERVICE_NOT_UP is returned rather than DCRPCER_SERVICE_TERMINATED.

(4) Relationship between error return values and synchronization point processing

The relationship between return values of the function dc_rpc_poll_any_replies() and synchronization point processing (commitment and rollback) is explained below. The description applies to the service request which is a transaction, rather than the service request which is not a transaction (including the case when DCRPC_TPNOTRAN is specified for flags of the function dc_rpc_call()).

- If commitment is performed even though the function dc_rpc_poll_any_replies() returns with an error

  The return value DCRPCER_TIMED_OUT may be returned due to abnormal termination of the service function which the service request is addressed, a node error, or network error. However, when the client UAP is not a transaction, the service function which the service request is addressed may terminate normally and database may be updated.

- Error return values which require rollback processing

  If the function dc_rpc_poll_any_replies() called from a transaction returns with an error, some return values always require rollback processing for the transaction (the server UAP enters in rollback_only state). In this case, rollback processing is always performed even if either of the commitment function or rollback function is used. The following return values of the function dc_rpc_poll_any_replies()

always require rollback processing for the transaction:

- DCRPCER_INVALID_REPLY
- DCRPCER_NO_BUFS_AT_SERVER
- DCRPCER_NO_SUCH_SERVICE
- DCRPCER_REPLY_TOO_BIG_RB

(5) When a response cannot be received by the function dc_rpc_poll_any_replies()

The function dc_rpc_poll_any_replies() cannot receive a response if either of the following functions is called by the UAP requesting a service with an asynchronous response-type RPC.

1. The receiving of asynchronous responses is rejected by the function dc_rpc_discard_further_replies()
2. Commitment or rollback processing is performed in the synchronization point processing function when a service is requested from a transaction.

The response returned after the above function is called is discarded. Receive all required asynchronous responses by using the function dc_rpc_poll_any_replies() before calling the above function when an asynchronous response-type RPC is used.

(6) Notes on using the function dc_rpc_poll_any_replies()

1. If the function dc_rpc_poll_any_replies() is called with the wait time as 0 (0 specified for the argument timeout), it may be probable that responses which have arrived cannot be received, because of a scheduling problem in a multithread environment. Note that a program which calls the function dc_rpc_poll_any_replies() with the wait time as 0 could be trapped in an endless loop until all responses are received.
2. If the function dc_rpc_poll_any_replies() with no descriptor specified returns with an error, the descriptor of the error response is undefined. To know the corresponding descriptor when the function dc_rpc_poll_any_replies() returns with an error, specify DCRPC_SPECIFIC_MSG for flags.

dc_rpc_service_retry - Retry a service function

Format

■ ANSI C, C++

```
#include <dcrpc.h>
int  dc_rpc_service_retry (void)
```

■ K&R C

```
#include <dcrpc.h>
int  dc_rpc_service_retry ()
```

Description

The function dc_rpc_service_retry() retries processing of the service function being executed. For a retry, call the function dc_rpc_service_retry() in the service function, then return the service function to be retried. After the return, the service function restarts in the same process.

If the service function called by a response RPC is retried, the values (the area to contain a response and the length of the response) set by the service function before the retry are invalidated.

If the function dc_rpc_service_retry is called after the number of retries set in the rpc_service_retry_count operand of the user service definition has been reached (including when 0 has been specified in the rpc_service_retry_count operand), the function returns error code DCRPCER_RETRY_COUNT_OVER, and control is returned due to the error. At this time, the service function is not retried. The service function called by a response RPC returns the contents of the area containing a response to the client UAP.

Return values

Return value	Return value (numeric)	Explanation
DC_OK	0	Normal termination.
DCRPCER_RETRY_COUNT_OVER	-377	The function dc_rpc_service_retry() is called more than the maximum number of service retries specified in the rpc_service_retry_count operand of the user service definition. The service function cannot be retried any more.

Return value	Return value (numeric)	Explanation
DCRPCER_PROTO	-302	The function dc_rpc_service_retry() is called under either of the following incorrect conditions: • The function is not called in the service function. • It is called within the global transaction.

Notes

1. Call the function dc_rpc_service_retry() under the following conditions. If these conditions are not satisfied, the function returns a value indicating an error, and control is returned.
   - The function dc_rpc_service_retry() is called in the service function.
   - The service function being executed is not within the global transaction.
2. The service function calling the function dc_rpc_service_retry() can reference the data passed by the client UAP, but cannot change it. If the contents of the input data area are changed, the system operation is undefined.
3. The function dc_rpc_service_retry() can be called only by the service function from which a service was requested by the OpenTP1 specific remote procedure call (function dc_rpc_call()). Processing of the other service functions cannot be retried by the function dc_rpc_service_retry().

dc_rpc_set_service_prio - Set a schedule priority of a service request

Format

- ANSI C, C++

```
#include <dcrpc.h>
void dc_rpc_set_service_prio (DCLONG prio)
```

- K&R C

```
#include <dcrpc.h>
void dc_rpc_set_service_prio (prio)
DCLONG      prio;
```

Description

The function `dc_rpc_set_service_prio()` sets a priority of a service request. It is called when controlling schedule priorities for individual service requests. The priority set by this function remains unchanged until it is updated by the same function. Therefore, if service requests are to be called at once with the same priority, call this function only once.

The priority set by this function will be reported to the server via the schedule queue by the function `dc_rpc_call()` which is called immediately after this function.

If this function is not called at all, the value 4, which is the default interpretation of the schedule service, is set as the priority of service requests.

Arguments whose value is set in the UAP

- `prio`

  Specify the schedule priority of the service request in the range from 0 to 8. This argument must always be set.

  The highest priority is represented by 1 and the lowest priority is represented by 8. If 0 is specified, the default interpretation of the schedule service will be in effect.

  If a value other than the above is specified, the function `dc_rpc_set_service_prio()` is ignored.

Return values

There is no return value of the function `dc_rpc_set_service_prio()`.

Notes

1. The priority specified for the service request is valid on a queue-receiving server

only when service_priority_control=Y (priority control in effect) is specified in the user service definition for the server UAP. If the server UAP to which the service request is addressed does not control priorities, this function is invalid even if called.

2. The function dc_rpc_set_service_prio() is invalid if it is called for a service request represented by the function dc_rpc_call() of the second or subsequent chained RPC or by the function dc_rpc_call() (DCNOFLAGS specified for flags) of synchronous-response-type RPC called to terminate the RPC chain.
3. The function dc_rpc_call() does not reset the service request priority to the default value. To reset the service request priority, recall the function dc_rpc_set_service_prio() with 0 specified for the argument prio.

Example

```
int     rc;
DCULONG in_len, len;
char    *buf;
/*  First service request:
 *  No priority is set (default interpretation of schedule service in effect)
 */
rc = dc_rpc_call("SPPG", "ECHO", "ex1", &in_len, buf, &len, DCNOFLAGS);
/*  Second service request: Priority = 8
 */
dc_rpc_set_service_prio(8);
rc = dc_rpc_call("SPPG", "ECHO", "ex2", &in_len, buf, &len, DCNOFLAGS);
/*  Third service request (chained RPC): Priority = 1
 */
dc_rpc_set_service_prio(1);
rc = dc_rpc_call("SPPG", "ECHO", "ex3", &in_len, buf, &len, DCRPC_CHINED);
      :
 (Chained RPC dc_rpc_call(DCRPC_CHAINED) repeated n times)
      :
rc = dc_rpc_call("SPPG", "ECHO", "ex3", &in_len, buf, &len,
                 DCNOFLAGS);
/*  (4 + n + 1)-th or subsequent service request:
 * Priority is reset (to default interpretation of schedule service)
 */
dc_rpc_set_service_prio(0);
rc = dc_rpc_call("SPPG", "ECHO", "ex4", &in_len, buf, &len, DCRPC_NOREPLY);
```

dc_rpc_set_watch_time - Update a service response waiting interval

Format

■ ANSI C, C++

```
#include <dcrpc.h>
int  dc_rpc_set_watch_time (int var)
```

■ K&R C

```
#include <dcrpc.h>
int  dc_rpc_set_watch_time (var)
int       var;
```

Description

The function `dc_rpc_set_watch_time()` change the response waiting interval of a service request. The value set by this function remains valid until the function `dc_rpc_close()` is called.

To reset the response waiting interval of a service request to the value which was in effect before this function was called, supply this function with the original value returned by the function `dc_rpc_get_watch_time()`.

This function does not change the value specified for the operand `watch_time` in the system common definition. The value set by this function influences only the function `dc_rpc_call()` which will be called later.

Arguments whose value is set in the UAP

■ `var`

Specify a new service response waiting interval in the range from 1 to 65535. For indefinite wait, specify 0.

Return values

Return value	Return value (numeric)	Explanation
`DC_OK`	`0`	Normal termination.
`DCRPCER_INVALID_ARGS`	`-301`	The value specified for `var` is invalid.
Other than the above.		An unprecedented error (e.g., program damage) occurred.

Real-time statistical information service (dc_rts_~)

This section gives the syntax and other information of the following real-time statistical information service functions:

- `dc_rts_utrace_put` - Acquire real-time statistical information for arbitrary section

dc_rts_utrace_put - Acquire real-time statistical information for arbitrary section

Format

- K&R C

```
#include <dcrts.h>
int dc_rts_utrace_put (event_id, flags);
DCLONG   event_id;
DCLONG   flags;
```

Description

The function `dc_rts_utrace_put()` acquires, as real-time statistical information, the execution time and execution count of the event set in `event_id` for arbitrary section within the UAP.

Arguments whose values are set in the UAP

- `event_id`

  Specify the event ID of the real-time statistical information to be acquired.

  The range of available event IDs is 1000000 to 2147483647.

- `flags`

  Set the processing to be executed by the function `dc_rts_utrace_put()`.

  `DCRTS_START`

  This flag starts measurement of the execution time of the event ID set in `event_id`.

  Real-time statistical information is not acquired when the function `dc_rts_utrace_put()` is called with this flag set.

  `DCRTS_END`

  This flag acquires the execution time of the event ID set in `event_id` and terminates measurement.

  `DCNOFLAGS`

  This flag acquires only the execution frequency of the event ID set in `event_id`. The execution time is 0 seconds.

Return values

Return value	Return value (numeric)	Explanation
DC_OK	0	Normal termination.
DCRTSER_PARAM	-7802	The value specified for an argument is incorrect.
DCRTSER_PROTO	-7803	The function dc_rpc_open() was not called.
		The function dc_rts_utrace_put() was called with an event ID for which measurement of the execution time has already started set in event_id and DCRTS_START set in flags.
		The function dc_rts_utrace_put() was called with an event ID for which measurement of the execution time has not yet started set in event_id and DCRTS_END set in flags.
DCRTSER_ITEM_OVER	-7804	The information cannot be acquired because the number of acquired items exceeds the value specified for the rts_item_max operand in the real-time statistical information service definition.
DCRTSER_ITEM_OVER_SRV	-7805	The information cannot be acquired because the number of acquired items per server exceeds the value specified for the rts_item_max operand in the real-time statistical information service definition. This return value indicates that statistical information for each service or for non-service processes has been acquired.
DCRTSER_ITEM_OVER_SVC	-7806	The information cannot be acquired because the number of acquired items per service or non-service process exceeds the value specified for the rts_item_max operand in the real-time statistical information service definition. This return value indicates that statistical information for each server has been acquired.
DCRTSER_NOMEM	-7807	Processing cannot be executed because process memory is insufficient.
DCRTSER_RTS_NOT_START	-7808	The real-time statistical information service has not started.
DCRTSER_NOENTRY	-7809	The caller of the function dc_rts_utrace_put has not been registered as a recipient for the acquisition of real-time statistical information on a server or service basis.

Return value	Return value (numeric)	Explanation
DCRTSER_VERSION	-7810	The UAP is linked with a library whose version is not supported by the currently operating real-time statistical information service.

Notes

1. The function dc_rts_utrace_put() cannot acquire real-time statistical information for the entire system.
2. On a UAP that uses a multi-server, if multiple processes simultaneously call the function dc_rts_utrace_put() with the same call source service and same event_id set, depending on the process, the function may not acquire statistical information. This is because lock is not performed during statistical information acquisition processing, and so write processes are performed simultaneously.
3. On a UAP that uses the XATMI interface, real-time statistical information cannot be acquired for individual services. Information is acquired as statistical information for all non-service processes.
4. The function dc_rts_utrace_put() cannot acquire a UAP trace.
5. This note applies after the function dc_rts_utrace_put called by specifying DCRTS_START in the flags argument returns DCRTSER_RTS_NOT_START or DCRTSER_NOENTRY. If the real-time statistical information service is started to add the calling UAP as a target of acquisition processing before the function dc_rts_utrace_put is called by specifying DCRTS_END in flags with the same event ID, the function returns DCRTSER_PROTO.

TAM file service (dc_tam_~)

This section gives the syntax and other information of the following TAM file service functions:

- `dc_tam_close` - Close a TAM table
- `dc_tam_delete` - Delete a TAM table record
- `dc_tam_get_inf` - Acquire TAM table status
- `dc_tam_open` - Open a TAM table
- `dc_tam_read` - Input a TAM table record
- `dc_tam_read_cancel` - Cancel the input of a TAM table record
- `dc_tam_rewrite` - Update a TAM table record on the assumption of input
- `dc_tam_status` - Acquire TAM table information
- `dc_tam_write` - Update/add a TAM table record

The functions for TAM file service (`dc_tam_~`) can be used only in UAPs of TP1/Server Base. They cannot be used in UAPs of TP1/LiNK.

dc_tam_close - Close a TAM table

Format

- ANSI C, C++

```
#include <dctam.h>
int  dc_tam_close (DCLONG tblid, DCLONG flags)
```

- K&R C

```
#include <dctam.h>
int dc_tam_close (tblid, flags)
DCLONG   tblid;
DCLONG   flags;
```

Description

The function `dc_tam_close()` closes a TAM table. After the function `dc_tam_close()` is called, the table descriptor specified for `tblid` cannot be used.

If the function `dc_tam_close()` returns with an error, all the resources acquired within this function are released, and the status before this function was called is regained.

If the function `dc_tam_open()` has been called outside the transaction, the function `dc_tam_close()` must also be called outside the transaction.

Similarly, if the function `dc_tam_open()` has been called inside the transaction, the function `dc_tam_close()` must also be called inside the transaction. If the function `dc_tam_close()` is not called before the transaction terminates, the TAM table is closed at the synchronization point.

If the function `dc_tam_close()` is called for the function `dc_tam_open()`, which was called outside the transaction, in the service function, terminate all the transactions in the same process which has accessed the TAM table to be closed. No error check is made on this termination. Operation is not ensured if the `dc_tam_close()` is called without the transactions terminated.

Arguments whose values are set in the UAP

- `tblid`

  Specify the table descriptor of the TAM table to be closed. This descriptor is the return value of the function `dc_tam_open()`.

- `flags`

  Specify `DCNOFLAGS`.

Return values

Return value	Return value (numeric)	Explanation
DC_OK	0	The TAM table was closed normally.
DCTAMER_PARAM_TID	-1700	The table descriptor specified for tblid is invalid.
DCTAMER_PARAM_FLG	-1708	The value specified for flags is invalid.
DCTAMER_TAMEND	-1720	The TAM service is being terminated.
DCTAMER_PROTO	-1721	The sequence of accessing the TAM table is invalid.
		The resource manager registration of the object file for transaction control having a linkage with the UAP is invalid. Alternatively, there is no linkage between the object file for transaction control and the UAP.
		atomic_update=N (nontransaction attribute) is specified in the user service definition of the UAP which called the function.
DCTAMER_TRNOPN	-1722	The function dc_tam_open() was called outside the transaction.
DCTAMER_NOOPEN	-1726	The TAM table is not open.
DCTAMER_MEMORY	-1769	The memory became insufficient.
DCTAMER_IO	-1770	An input/output error occurred.

dc_tam_delete - Delete a TAM table record

Format

- ANSI C, C++

```
#include <dctam.h>
int  dc_tam_delete (DCLONG tblid, struct DC_TAMKEY *keyadr,
                    int keyno, char *bufadr, int bufsize,
                    DCLONG flags)
```

- K&R C

```
#include <dctam.h>
int       dc_tam_delete (tblid, keyadr, keyno, bufadr, bufsize,
                         flags)
DCLONG     tblid;
struct  DC_TAMKEY *keyadr;
int        keyno;
char       *bufadr;
int        bufsize;
DCLONG     flags;
```

Description

The function `dc_tam_delete()` deletes a record indicated as a key value from a TAM table. The record to be deleted can be saved in the buffer. However, if the function `dc_tam_delete()` returns with an error, the buffer contents cannot be ensured.

If a TAM table is open under lock in records, lock in tables can be enabled with lock for update processing.

If the function `dc_tam_delete()` returns with an error, all the resources specified in this function are released, and the status before this function was called is regained. However, if an attempt is made to delete a TAM table which was acquired under lock for reference processing before this function was called, lock for update processing is enabled. (Lock for reference processing is not regained.)

Note the following when multiple records are specified for deletion:

- Even if one of the records causes an error, the processing of all the records specified in the function `dc_tam_delete()` results in an error, and the status before this function was called is regained.

Arguments whose values are set in the UAP

- `tblid`

  Specify the table descriptor of the TAM table from which a record is deleted. This

descriptor is the return value of the function dc_tam_open().

- keyadr

  Specify the address of the structure having the key value address of the record to be deleted. The structure format is as follows:

```
struct DC_TAMKEY {
                  char *keyname;
                  };
```

  - keyname

    Specify the address of the key value. The key value must be specified with the length of the key area of the record to be deleted.

- keyno

  Specify the number of request records (number of structures specified for keyadr).

- bufadr

  If the record to be deleted is saved in the buffer, specify the buffer address. If DCTAM_NOOUTREC (the record to be deleted is not saved) is specified for flags, the specification for this argument is invalidated.

- bufsize

  If the record to be deleted is saved in the buffer, specify the length of the buffer. The return buffer length must be equal to or greater than (record length x number of request records). If DCTAM_NOOUTREC (the record to be deleted is not saved) is specified for flags, the specification for this argument is invalidated.

- flags

  Specify the following items in the format shown below:

  - Record access type
  - Lock release wait type

```
{DCTAM_NOOUTREC|DCTAM_OUTREC}[|{DCTAM_WAIT|DCTAM_NOWAIT}]
```

  - Flag 1

    Do not omit the specification of record access type. More than one access type cannot be specified at a time.

    DCTAM_NOOUTREC

    The record to be deleted is not saved.

    DCTAM_OUTREC

The record to be deleted is saved.

- Flag 2

  If no lock release wait type is specified, the function does not wait for the resource to be released from lock and returns with an error. More than one lock release wait type cannot be specified at a time.

  DCTAM_WAIT

  The function waits for the resource to be released from lock.

  DCTAM_NOWAIT

  The function does not wait for the resource to be released from lock and returns with an error.

Return values

Return value	Return value (numeric)	Explanation
DC_OK	0	The record was deleted from the TAM table normally.
DCTAMER_PARAM_TID	-1700	The table identifier specified for tblid is invalid.
DCTAMER_PARAM_KEY	-1702	The key value specified for keyadr is invalid.
DCTAMER_PARAM_KNO	-1703	The value specified for keyno is invalid.
DCTAMER_PARAM_BFA	-1704	The value specified for bufadr is invalid.
DCTAMER_PARAM_BFS	-1705	The buffer length specified for bufsize is too short.
DCTAMER_PARAM_FLG	-1708	The value specified for flags is invalid.
DCTAMER_NOTTAM	-1709	The table specified for tblid is not a TAM table.
DCTAMER_TAMEND	-1720	The TAM service is being terminated.
DCTAMER_PROTO	-1721	The sequence of accessing the TAM table is invalid.
		The resource manager registration of the object file for transaction control having a linkage with the UAP is invalid. Alternatively, there is no linkage between the object file for transaction control and the UAP.
		atomic_update=N (nontransaction attribute) is specified in the user service definition of the UAP which called the function.
DCTAMER_RMTBL	-1723	The TAM table was deleted.
DCTAMER_NOLOAD	-1724	The TAM table was not loaded.

Return value	Return value (numeric)	Explanation
DCTAMER_NOOPEN	-1726	The TAM table is not open.
DCTAMER_LOGHLD	-1727	The TAM table is in logical shutdown state.
DCTAMER_OBSHLD	-1728	The TAM table is in shutdown state due to an error.
DCTAMER_ACSATL	-1730	Execution is impossible in the access mode of the TAM table specified in the TAM service definition.
DCTAMER_NOREC	-1731	The specified record does not exist.
DCTAMER_LOCK	-1736	A lock error occurred. If DCTAM_WAIT is specified for flags, the resource could not be acquired because a timeout occurred (the wait time specified in the lock service definition was exceeded).
DCTAMER_DLOCK	-1737	A deadlock occurred.
DCTAMER_TBLVR	-1760	The version of the TAM library linked to the UAP does not allow the UAP to operate with the current TAM table.
DCTAMER_FLSVR	-1761	The version of the TAM library linked to the UAP does not allow the UAP to operate with the current OpenTP1 file service.
DCTAMER_RECOBS	-1764	The record has been damaged.
DCTAMER_TRNNUM	-1765	The number of transactions exceeds the maximum number of transactions which can be managed by the TAM service.
DCTAMER_OPENNUM	-1766	The number of open character special files exceeds the specified limit.
DCTAMER_ACCESSS	-1767	The access permission for special files has not been granted.
DCTAMER_ACCESSF	-1768	The access permission for TAM files has not been granted.
DCTAMER_MEMORY	-1769	The memory became insufficient.
DCTAMER_IO	-1770	An input/output error occurred.
DCTAMER_TMERR	-1771	A transaction service error occurred.
DCTAMER_ACCESS	-1773	The TAM file to be accessed is protected with the security facility. The UAP that called the function dc_tam_delete() has no access permission.

Notes

To delete all records stored in the hash format TAM table:

1. Save the key value of the record found by first retrieval as *variable-1*.
2. Using the key value of *variable-1*, execute a NEXT retrieval.
3. Save the key value of the record found in step 2 as *variable-2*.
4. Delete the key value record that was saved as *variable-1*.
5. Save the key value of *variable-2* as *variable-1*.
6. Repeat steps 2 to 5 until step 2 encounters an error (NEXT retrieval).
7. After step 2 has encountered an error, delete the key value record that was last saved as *variable-1*.

Note that when you delete all records, performing the following steps may exert a high load on the CPU.

1. Execute a first retrieval of records.
2. Delete the record found in step 1.
3. Repeat step 1 and step 2 (that is, continue executing a first retrieval of records and deleting the record found).

dc_tam_get_inf - Acquire TAM table status

Format

- ANSI C, C++

```
#include <dctam.h>
int  dc_tam_get_inf (char *tblname, DCLONG flags)
```

- K&R C

```
#include <dctam.h>
int  dc_tam_get_inf (tblname, flags)
char    *tblname;
DCLONG  flags;
```

Description

The function `dc_tam_get_inf()` acquires the status of a TAM table. The TAM table status to be acquired includes the following:

- Open state
- Closed state
- Logical shutdown state
- Shutdown state due to an error

The function `dc_tam_get_inf()` can be called both outside and inside the transaction.

The function `dc_tam_get_inf()` returns assuming that the specified TAM table is open in the following case:

- The function `dc_tam_open()` is not called from the process that called the function `dc_tam_get_inf()`, but another process has called the function `dc_tam_open()` for the specified TAM table.

Arguments whose values are set in the UAP

- `tblname`

  Specify the address of the name of the TAM table whose status is to be acquired. The TAM table can be specified with up to 32 characters. The character string must end with a null character.

- `flags`

  Specify `DCNOFLAGS`.

Return values

With a positive return value (indicating the TAM table status)

Return value	Return value (numeric)	Explanation
DCTAM_STS_OPN	1	The TAM table is open.
DCTAM_STS_CLS	2	The TAM table is closed.
DCTAM_STS_LHLD	3	The TAM table is in logical shutdown state.
DCTAM_STS_OHLD	4	The TAM table is in shutdown state due to an error.

With a negative return value (indicating that an error occurred)

Return value	Return value (numeric)	Explanation
DCTAMER_PARAM_TBL	-1701	The value specified for tblname is invalid.
DCTAMER_PARAM_FLG	-1708	The value specified for flags is invalid.
DCTAMER_UNDEF	-1710	The TAM table has not been defined.
DCTAMER_TAMEND	-1720	The TAM service is being terminated.
DCTAMER_PROTO	-1721	The sequence of accessing the TAM table is invalid.
		The resource manager registration of the object file for transaction control having a linkage with the UAP is invalid. Alternatively, there is no linkage between the object file for transaction control and the UAP.
		atomic_update=N (nontransaction attribute) is specified in the user service definition of the UAP which called the function.
DCTAMER_TAMVR	-1762	The version of the TAM library linked to the UAP does not allow the UAP to operate with the current TAM service.
DCTAMER_ACCESS	-1773	A TAM file to be accessed is protected with the security facility. The UAP that called the function dc_tam_get_inf() has no access permission.
DCTAMER_NO_ACL	-1772	A TAM file to be accessed is protected with the security facility. There is no ACL for the corresponding file.

dc_tam_open - Open a TAM table

Format

- ANSI C, C++

```
#include <dctam.h>
DCLONG dc_tam_open (char *tblname, DCLONG flags)
```

- K&R C

```
#include <dctam.h>
DCLONG dc_tam_open (tblname, flags)
char      *tblname;
DCLONG    flags;
```

Description

The function `dc_tam_open()` opens a TAM table. The function `dc_tam_open()` can be called both outside and inside the transaction.

If the function `dc_tam_open()` is called inside the transaction and lock in tables is specified as a lock type, lock in tables is enabled with lock for update processing.

If the function `dc_tam_open()` returns with an error, all the resources acquired within this function are released, and the status before this function was called is regained.

Arguments whose values are set in the UAP

- `tblname`

  Specify the name of the TAM table to be opened. The TAM table name can be specified with up to 32 characters. The character string must end with a null character.

- `flags`

  Specify whether to enable lock in tables or in records in the format shown below.

```
[{DCTAM_TBL_EXCLUSIVE[|{DCTAM_WAIT|DCTAM_NOWAIT}]|
DCTAM_REC_EXCLUSIVE}]
```

  - Flag 1

    Lock in tables is enabled with lock for update processing. Lock in records is enabled within the record access function.

    More than one lock release wait type cannot be specified at a time. If the function `dc_tam_open()` is called outside the transaction, lock in tables cannot be specified.

The default is DCTAM_REC_EXCLUSIVE.

DCTAM_TBL_EXCLUSIVE

Lock in tables

DCTAM_REC_EXCLUSIVE

Lock in records

- Flag 2

  When lock in tables is specified, specify a lock release wait type if competition for a resource occurs. More than one lock release type cannot be specified at a time.

The default is DCTAM_NOWAIT.

DCTAM_WAIT

The function waits for the resource to be released from lock.

DCTAM_NOWAIT

The function does not wait for the resource to be released from lock, and returns with an error

The table below shows the correspondence between flag values specified for flags and the specified type of lock.

Flag 1[#1]	Flag 2[#2]	Lock specified for flags
TBL_EXCLUSIVE	WAIT	Lock in tables, and waiting for release from lock if a lock error occurs
	NOWAIT	Lock in tables, and error return if a lock error occurs
REC_EXCLUSIVE	N/A	Lock in records

Legend:

N/A: Cannot be specified.

#1: The default is REC_EXCLUSIVE.

#2: The default is NOWAIT.

Return values

Return value	Return value (numeric)	Explanation
Positive integer		The positive integer indicates the table descriptor.
DCTAMER_PARAM_TBL	-1701	The value specified for tblname is invalid.

Return value	Return value (numeric)	Explanation
DCTAMER_PARAM_FLG	-1708	The value specified for flags is invalid.
DCTAMER_NOTTAM	-1709	The table specified for tblname is not a TAM table.
DCTAMER_UNDEF	-1710	The TAM table has not been defined.
DCTAMER_TAMEND	-1720	The TAM service is being terminated.
DCTAMER_PROTO	-1721	The sequence of accessing the TAM table is invalid.
		The resource manager registration of the object file for transaction control having a linkage with the UAP is invalid. Alternatively, there is no linkage between the object file for transaction control and the UAP.
		atomic_update=N (nontransaction attribute) is specified in the user service definition of the UAP which called the function.
DCTAMER_NOLOAD	-1724	The TAM table was not loaded.
DCTAMER_OPENED	-1725	The TAM table is open.
DCTAMER_LOGHLD	-1727	The TAM table is in logical shutdown state.
DCTAMER_OBSHLD	-1728	The TAM table is in shutdown state due to an error.
DCTAMER_LOCK	-1736	A lock error occurred. If DCTAM_WAIT is specified for flags, the resource could not be acquired because a timeout occurred (the wait time specified in the lock service definition was exceeded).
DCTAMER_DLOCK	-1737	A deadlock occurred.
DCTAMER_TBLVR	-1760	The version of the TAM library linked to the UAP does not allow the UAP to operate with the current TAM table.
DCTAMER_FLSVR	-1761	The version of the TAM library linked to the UAP does not allow the UAP to operate with the current OpenTP1 file service.
DCTAMER_TAMVR	-1762	The version of the TAM library linked to the UAP does not allow the UAP to operate with the current TAM service.
DCTAMER_RECOBS	-1764	The record has been damaged.

Return value	Return value (numeric)	Explanation
DCTAMER_TRNNUM	-1765	The number of transactions exceeds the maximum number of transactions which can be managed by the TAM service.
DCTAMER_OPENNUM	-1766	The number of open character special files exceeds the specified limit.
DCTAMER_ACCESSS	-1767	The access permission for special files has not been granted.
DCTAMER_ACCESSF	-1768	The access permission for TAM files has not been granted.
DCTAMER_MEMORY	-1769	The memory became insufficient.
DCTAMER_IO	-1770	An input/output error occurred.
DCTAMER_TMERR	-1771	A transaction service error occurred.
DCTAMER_NO_ACL	-1772	A TAM file to be opened is protected with the security facility. There is no ACL for the corresponding file.

dc_tam_read - Input a TAM table record

Format

■ ANSI C, C++

```
#include <dctam.h>
int  dc_tam_read (DCLONG tblid, struct DC_TAMKEY *keyadr,
                  int keyno, char *bufadr, int bufsize,
                  DCLONG flags)
```

■ K&R C

```
#include <dctam.h>
int       dc_tam_read (tblid, keyadr, keyno, bufadr, bufsize,
                       flags)
DCLONG    tblid;
struct DC_TAMKEY   *keyadr;
int       keyno;
char      *bufadr;
int       bufsize;
DCLONG    flags;
```

Description

According to the search type specified for `flags`, the function `dc_tam_read()` inputs a TAM table record for reference or update processing. The table below shows the relationship between search types and index types.

Table 2-2: Relationship between search types and index types

Search type	Outline of search processing	
	Index type: hash format	Index type: tree format
`'key-value='search`	The record having the specified key value is searched for. If the record having the specified key value is not found, an error is returned.	The record having the specified key value is searched for. If the record having the specified key value is not found, an error is returned.
`'key-value<='search`	An error is returned.	The record having a key value equal to or greater than the specified key value is searched for.
`'key-value<'search`	An error is returned.	The record having a key value greater than the specified key value is searched for.
`'key-value>='search`	An error is returned.	The record having a key value equal to or smaller than the specified key value or less is searched for.

Search type	Outline of search processing	
	Index type: hash format	Index type: tree format
'key-value>'search	An error is returned.	The record having a key value smaller than the specified key value is searched for.
First record search#	The first record that was hashed in correspondence with the key value is searched for. The key value specified for keyadr is ignored.	An error is returned.
NEXT search#	The next record that was hashed in correspondence with the key value is searched for.	An error is returned.

#: All the records in the TAM table can be searched for by using the first record search and NEXT search in the following conditions:

- The hash format is specified as the index type.
- When a TAM table file is created, a key value is assigned to the data part (the -s option not specified in the tamcre command).

If lock is specified with input for reference processing, lock in tables and lock in records are enabled with lock for reference processing. If a TAM table open under lock in records is input for update processing, lock in tables is enabled with lock for reference processing, and lock in records is enabled with lock for update processing.

If the function dc_tam_read() returns with an error, all the resources specified in this function are released, and the status before this function was called is regained. However, if a record which was acquired under lock for reference processing before this function was called is input for update processing, lock for update processing is enabled. (Lock for reference processing is not regained.) If an error is returned, the buffer contents cannot be ensured.

Note the following when multiple records are specified for input:

- Even if one of the records causes an error, the processing of all the records specified in the function dc_tam_read() results in an error.

Arguments whose values are set in the UAP

- tblid

  Specify the table descriptor of the TAM table to which a record is to be input. The table descriptor is the value returned with the function dc_tam_open().

- keyadr

  Specify the address of the structure having the key value address for searching for the record. The structure format is as follows:

```
struct DC_TAMKEY {
                 char *keyname;
                 };
```

- keyname

  Specify the address of the key value. The key value must be specified with the length of the key area of the record to be input.

- keyno

  Specify the number of request records (number of structures specified for keyadr).

- bufadr

  Specify the address of the buffer to which the record is to be input.

- bufsize

  Specify the length of the buffer to which the record is to be input. The return buffer length must be equal to or greater than (record length x number of request records).

- flags

  Specify the following items in the format shown below:

  - Record search type
  - Record access type
  - Whether to enable lock for reference processing when the lock is specified (lock enabled/disabled type)
  - Lock release wait type if competition for a resource occurs

```
{Flag 1}
|{DCTAM_REFERENCE[|{DCTAM_EXCLUSIVE|DCTAM_NOEXCLUSIVE}]
|DCTAM_MODIFY}[|{DCTAM_WAIT|DCTAM_NOWAIT}]
```

  - Flag 1

    The specification of record search type cannot be omitted. More than one record search type cannot be specified at a time.

    DCTAM_EQLSRC: 'key-value=' is searched for. (Hash and tree formats)

    DCTAM_GRTEQLSRC: 'key-value<=' is searched for. (Tree format)

    DCTAM_GRTSRC: 'key-value<' is searched for. (Tree format)

    DCTAM_LSSEQLSRC: 'key-value>=' is searched for. (Tree format)

    DCTAM_LSSSRC: 'key-value>' is searched for. (Tree format)

    DCTAM_FIRSTSRC: Search processing starts from the first record. (Hash format)

DCTAM_NEXTSRC: Search processing starts from the record following the specified key value. (Hash format)

- Flag 2

  The specification of record access type cannot also be omitted. More than one access type cannot also be specified at a time.

  DCTAM_REFERENCE: Lock for reference processing

  DCTAM_MODIFY: Lock for update processing

- Flag 3

  If lock for reference processing is specified, also specify whether to enable the lock. More than one lock enabled/disabled type cannot be specified at a time. The default is DCTAM_NOEXCLUSIVE.

  DCTAM_EXCLUSIVE: Lock is enabled.

  DCTAM_NOEXCLUSIVE: Lock is disabled.

- Flag 4

  More than one lock release wait type cannot also be specified. The default is DCTAM_NOWAIT.

  DCTAM_WAIT: The function waits for the resource to be released from lock.

  DCTAM_NOWAIT: The function does not wait for the resource to be released from lock, and returns with an error.

The table below shows the correspondence between flag values specified for flags and the specified type of lock.

Flag 1	Flag 2	Flag 3#1	Flag 4#2	Lock specified for flags
EQLSRC GRTEQLSRC GRTSRC LSSEQLSRC LSSSRC FIRSTSRCN EXTSRC	REFERENCE	EXCLUSIVE	WAIT	Input for reference, lock used, and waiting for release from lock if a lock error occurs
			NOWAIT	Input for reference, lock used, and error return if a lock error occurs
		NOEXCLUSIVE	N/A	Input for reference, and lock not used
	MODIFY	--	WAIT	Input for update, and waiting for release from lock if a lock error occurs
			NOWAIT	Input for update, and error return if a lock error occurs

Legend:

N/A: Cannot be specified.

--: Specify always EXCLUSIVE. NOEXCLUSIVE cannot be specified.

#1: The default is NOEXCLUSIVE.

#2: The default is NOWAIT.

Return values

Return value	Return value (numeric)	Explanation
DC_OK	0	The TAM table record was input normally.
DCTAMER_PARAM_TID	-1700	The table descriptor specified for tblid is invalid.
DCTAMER_PARAM_KEY	-1702	The key value specified for keyadr is invalid.
DCTAMER_PARAM_KNO	-1703	The value specified for keyno is invalid.
DCTAMER_PARAM_BFA	-1704	The value specified for bufadr is invalid.
DCTAMER_PARAM_BFS	-1705	The buffer length specified for bufsize is too short.
DCTAMER_PARAM_FLG	-1708	The value specified for flags is invalid.
DCTAMER_NOTTAM	-1709	The table specified for tblid is not a TAM table.
DCTAMER_TAMEND	-1720	The TAM service is being terminated.

Return value	Return value (numeric)	Explanation
DCTAMER_PROTO	-1721	The sequence of accessing the TAM table is invalid.
		The resource manager registration of the object file for transaction control having a linkage with the UAP is invalid. Alternatively, there is no linkage between the object file for transaction control and the UAP.
		atomic_update=N (nontransaction attribute) is specified in the user service definition of the UAP which called the function.
DCTAMER_RMTBL	-1723	The TAM table was deleted.
DCTAMER_NOLOAD	-1724	The TAM table was not loaded.
DCTAMER_NOOPEN	-1726	The TAM table is not open.
DCTAMER_LOGHLD	-1727	The TAM table is in logical shutdown state.
DCTAMER_OBSHLD	-1728	The TAM table is in shutdown state due to an error.
DCTAMER_IDXTYP	-1729	Execution is impossible with the index type of the TAM table specified for creation of a TAM table file.
DCTAMER_ACSATL	-1730	Execution is impossible in the access mode of the TAM table specified in the TAM service definition.
DCTAMER_NOREC	-1731	A record satisfying the search conditions specified for flags is not found.
DCTAMER_LOCK	-1736	A lock error occurred. If DCTAM_WAIT is specified for flags, the resource could not be acquired because a timeout occurred (the wait time specified in the lock service definition was exceeded).
DCTAMER_DLOCK	-1737	A deadlock occurred.
DCTAMER_TBLVR	-1760	The version of the TAM library linked to the UAP does not allow the UAP to operate with the current TAM table.
DCTAMER_FLSVR	-1761	The version of the TAM library linked to the UAP does not allow the UAP to operate with the current OpenTP1 file service.
DCTAMER_RECOBS	-1764	The record has been damaged.
DCTAMER_TRNNUM	-1765	The number of transactions exceeds the maximum number of transactions which can be managed by the TAM service.

Return value	Return value (numeric)	Explanation
DCTAMER_OPENNUM	-1766	The number of open character special files exceeds the specified limit.
DCTAMER_ACCESSS	-1767	The access permission for special files has not been granted.
DCTAMER_ACCESSF	-1768	The access permission for TAM files has not been granted.
DCTAMER_MEMORY	-1769	The memory became insufficient.
DCTAMER_IO	-1770	An input/output error occurred.
DCTAMER_TMERR	-1771	A transaction service error occurred.
DCTAMER_ACCESS	-1773	A TAM file to be accessed is protected with the security facility. The UAP that called the function dc_tam_read() has no access permission.

dc_tam_read_cancel - Cancel the input of a TAM table record

Format

- ANSI C, C++

```
#include <dctam.h>
int  dc_tam_read_cancel (DCLONG tblid, struct DC_TAMKEY
                         *keyadr, int keyno, DCLONG flags)
```

- K&R C

```
#include <dctam.h>
int dc_tam_read_cancel (tblid, keyadr, keyno, flags)
DCLONG           tblid;
struct DC_TAMKEY *keyadr;
int              keyno;
DCLONG           flags;
```

Description

The function `dc_tam_read_cancel()` cancels the input for reference or update processing with lock specified in the function `dc_tam_read()`, and resets lock in records.

For an updated or added record, the input for reference processing with lock specified cannot be canceled. For a record updated by the function `dc_tam_rewrite()`, the input for update processing cannot also be canceled.

If the input for update processing is canceled for updated/added records or for the records of a TAM table open under lock in tables, lock is not reset.

After the function `dc_tam_read_cancel()` cancels input, other transactions are not allowed to add/delete a record to/from the input TAM table until the transaction terminates.

If the function `dc_tam_read_cancel()` returns with an error, all the resources acquired within this function are released, and the status before this function was called is regained. When a request is made to access multiple specified records, even if one of the records causes an error, processing is stopped and an error is returned.

Arguments whose values are set in the UAP

- `tblid`

  Specify the table descriptor of the TAM table whose record input is to be canceled. The table descriptor is the value returned with the function `dc_tam_open()`.

- keyadr

  Specify the address of the structure having the address of the key value of the record whose input is to be canceled. The structure format is as follows:

```
struct DC_TAMKEY {
                  char *keyname;
                 };
```

  - keyname

    Specify the address of the key value. The key value must be specified with the length of the key area of the record whose input is to be canceled.

- keyno

  Specify the number of request records (number of structures specified for keyadr).

- flags

  Specify DCNOFLAGS.

Return values

Return value	Return value (numeric)	Explanation
DC_OK	0	Search for the TAM table record was canceled, and lock in records was reset normally.
DCTAMER_PARAM_TID	-1700	The table descriptor specified for tblid is invalid.
DCTAMER_PARAM_KEY	-1702	The key value specified for keyadr is invalid.
DCTAMER_PARAM_KNO	-1703	The value specified for keyno is invalid.
DCTAMER_PARAM_FLG	-1708	The value specified for flags is invalid.
DCTAMER_NOTTAM	-1709	The table specified for tblid is not a TAM table.
DCTAMER_TAMEND	-1720	The TAM service is being terminated.
DCTAMER_PROTO	-1721	The sequence of accessing the TAM table is invalid.
		The resource manager registration of the object file for transaction control having a linkage with the UAP is invalid. Alternatively, there is no linkage between the object file for transaction control and the UAP.
		atomic_update=N (nontransaction attribute) is specified in the user service definition of the UAP which called the function.

Return value	Return value (numeric)	Explanation
DCTAMER_RMTBL	-1723	The TAM table was deleted.
DCTAMER_NOLOAD	-1724	The TAM table was not loaded.
DCTAMER_NOOPEN	-1726	The TAM table is not open.
DCTAMER_LOGHLD	-1727	The TAM table is in logical shutdown state.
DCTAMER_OBSHLD	-1728	The TAM table is in shutdown state due to an error.
DCTAMER_NOREC	-1731	The specified record does not exist.
DCTAMER_SEQENCE	-1732	The function dc_tam_read() was not called.
DCTAMER_EXWRITE	-1733	The table identifier specified for tblid indicates the record updated or added by the function dc_tam_write().
DCTAMER_EXREWRT	-1734	The table descriptor specified for tblid was updated by the function dc_tam_rewrite().
DCTAMER_TBLVR	-1760	The version of the TAM library linked to the UAP does not allow the UAP to operate with the current TAM table.
DCTAMER_FLSVR	-1761	The version of the TAM library linked to the UAP does not allow the UAP to operate with the current OpenTP1 file service.
DCTAMER_TRNNUM	-1765	The number of transactions exceeds the maximum number of transactions which can be managed by the TAM service.
DCTAMER_OPENNUM	-1766	The number of open character special files exceeds the specified limit.
DCTAMER_ACCESSS	-1767	The access permission for special files has not been granted.
DCTAMER_ACCESSF	-1768	The access permission for TAM table files has not been granted.
DCTAMER_MEMORY	-1769	The memory became insufficient.
DCTAMER_IO	-1770	An input/output error occurred.
DCTAMER_TMERR	-1771	A transaction service error occurred.

dc_tam_rewrite - Update a TAM table record on the assumption of input

Format

- ANSI C, C

```
#include <dctam.h>
int  dc_tam_rewrite (DCLONG tblid, struct DC_TAMKEY
                     *keyadr, int keyno, char *datadr,
                     int datsize, DCLONG flags)
```

- K&R C

```
#include <dctam.h>
int  dc_tam_rewrite (tblid, keyadr, keyno, datadr, datsize,
                     flags)
DCLONG    tblid;
struct DC_TAMKEY *keyadr;
int       keyno;
char      *datadr;
int       datsize;
DCLONG    flags;
```

Description

The function `dc_tam_rewrite()` updates and outputs a record input by the function `dc_tam_read`.

Once the function `dc_tam_read()` is called to input a record for update processing, the function `dc_tam_rewrite()` can be called any number of times before the synchronization point of the transaction is acquired. However, the function `dc_tam_rewrite()` cannot be called after the function `dc_tam_delete()` or `dc_tam_read_cancel()`.

If the function `dc_tam_rewrite()` returns with an error, all the resources specified within this function are released, and the status before this function was called is regained.

When a request is made to update multiple specified records, even if one of the records causes an error, the processing of all the records specified in this function results in an error.

The key value storage location in the update data and the key area length are as specified in the `tamcre` command used for creation of a TAM table file.

The data part has a key value if the key value is assigned to the data part (the `-s` option not specified in the `tamcre` command) when a TAM table file is created. Therefore, an error is returned if the key value specified in the function `dc_tam_rewrite()` is

not found in the update data. The data part has no key value if no key value is assigned to the data part (the `-s` option specified in the `tamcre` command). In this case, no check is made on the contents of the update data.

Arguments whose values are set in the UAP

- `tblid`

  Specify the table descriptor of the TAM table whose record is to be updated. The table descriptor is the value returned with the function `dc_tam_open()`.

- `keyadr`

  Specify the address of the structure having the address of the key value of the record to be updated. The structure format is as follows:

```
struct DC_TAMKEY {
                  char *keyname;
                 };
```

  - `keyname`

    Specify the address of the key value. The key value must be specified with the length of the key area of the record to be updated.

- `keyno`

  Specify the number of request records (number of structures specified for `keyadr`).

- `datadr`

  Specify the address of the update data.

- `datsize`

  Specify the length of the update data. The update data length must be equal to or greater than (record length x number of request records).

- `flags`

  Specify `DCNOFLAGS`.

Return values

Return value	Return value (numeric)	Explanation
`DC_OK`	`0`	The TAM table record was updated normally.
`DCTAMER_PARAM_TID`	`-1700`	The table descriptor specified for `tblid` is invalid.
`DCTAMER_PARAM_KEY`	`-1702`	The key value specified for `keyadr` is invalid.
`DCTAMER_PARAM_KNO`	`-1703`	The value specified for `keyno` is invalid.

Return value	Return value (numeric)	Explanation
DCTAMER_PARAM_DTA	-1706	The value specified for datadr is invalid.
DCTAMER_PARAM_DTS	-1707	The data length specified for datsize is too short.
DCTAMER_PARAM_FLG	-1708	The value specified for flags is invalid.
DCTAMER_NOTTAM	-1709	The table specified for tblid is not a TAM table.
DCTAMER_TAMEND	-1720	The TAM service is being terminated.
DCTAMER_PROTO	-1721	The sequence of accessing the TAM table is invalid.
		The resource manager registration of the object file for transaction control having a linkage with the UAP is invalid. Alternatively, there is no linkage between the object file for transaction control and the UAP.
		atomic_update=N (nontransaction attribute) is specified in the user service definition of the UAP which called the function.
DCTAMER_RMTBL	-1723	The TAM table was deleted.
DCTAMER_NOLOAD	-1724	The TAM table was not loaded.
DCTAMER_NOOPEN	-1726	The TAM table is not open.
DCTAMER_LOGHLD	-1727	The TAM table is in logical shutdown state.
DCTAMER_OBSHLD	-1728	The TAM table is in shutdown state due to an error.
DCTAMER_NOREC	-1731	The specified record does not exist.
DCTAMER_SEQENCE	-1732	The function dc_tam_read() was not called.
DCTAMER_TBLVR	-1760	The version of the TAM library linked to the UAP does not allow the UAP to operate with the current TAM table.
DCTAMER_FLSVR	-1761	The version of the TAM library linked to the UAP does not allow the UAP to operate with the current OpenTP1 file service.
DCTAMER_RECOBS	-1764	The record has been damaged.
DCTAMER_TRNNUM	-1765	The number of transactions exceeds the maximum number of transactions which can be managed by the TAM service.

Return value	Return value (numeric)	Explanation
DCTAMER_OPENNUM	-1766	The number of open character special files exceeds the specified limit.
DCTAMER_ACCESSS	-1767	The access permission for special files has not been granted.
DCTAMER_ACCESSF	-1768	The access permission for TAM files has not been granted.
DCTAMER_MEMORY	-1769	The memory became insufficient.
DCTAMER_IO	-1770	An input/output error occurred.
DCTAMER_TMERR	-1771	A transaction service error occurred.

dc_tam_status - Acquire TAM table information

Format

- ANSI C, C++

```
#include <dctam.h>
int  dc_tam_status (char *tblname, struct DC_TAMSTAT
                    *stbuf, DCLONG flags)
```

- K&R C

```
#include <dctam.h>
int  dc_tam_status (tblname, stbuf, flags)
char       *tblname;
struct  DC_TAMSTAT  *stbuf;
DCLONG     flags;
```

Description

The function `dc_tam_status()` returns TAM table information in a structure `DC_TAMSTAT`. The following values are returned by the function:

- TAM file name
- TAM table status
- Number of records in use
- Maximum number of records
- Index type
- Access type
- Loading opportunity
- TAM record length
- Key length
- Key start position
- Security attribute

Arguments whose value is set in the UAP

- `tblname`

  Specify the name of the TAM table from which information is acquired up to 32 characters. End the character string with a null character.

- stbuf

  Specify the address of a structure DC_TAMSTAT that receives TAM table information. The TAM table status set in the function dc_tam_status() is returned in the structure.

- flags

  Specify DCNOFLAGS.

Argument whose value is returned from OpenTP1

- stbuf

  TAM table information is returned in the format of structure DC_TAMSTAT as follows:

```
struct  DC_TAMSTAT {
                    char    st_file_name[64];
                    DCLONG  st_tbl_stat;
                    DCLONG  st_rec_usenum;
                    DCLONG  st_tbl_maxnum;
                    char    st_idx_type;
                    char    st_acs_type;
                    char    st_lod_type;
                    char    reserve1;
                    DCLONG  st_rec_len;
                    DCLONG  st_key_len;
                    DCLONG  st_key_pos;
                    DCLONG  st_tbl_sec;
                    DCLONG  reserve2[8];
                    };
```

  - st_file_name

    The TAM file name is returned.

  - st_tbl_stat

    The TAM table status is returned as follows:

    DCTAM_STS_OPN: The TAM table is opened.

    DCTAM_STS_CLS: The TAM table is closed.

    DCTAM_STS_LHLD: The TAM table is in logical shutdown state.

    DCTAM_STS_OHLD: The TAM table is in shutdown state due to an error.

  - st_rec_usenum

    The number of records currently used in the TAM table is returned. However, this value is not assured if a record is added or deleted after the function dc_tam_status() is called.

  - st_tbl_maxnum

The maximum number of records for the TAM table is returned.

- `st_idx_type`

  The index type of the TAM table is returned as follows:

  `DCTAM_STS_HASH`: The TAM table adopts hash format.

  `DCTAM_STS_TREE`: The TAM table adopts tree format.

- `st_acs_type`

  The access type of the TAM table is returned as follows:

  `DCTAM_STS_READ`: The TAM table is reference-only type.

  `DCTAM_STS_REWRITE`: The TAM table is overwrite type (any record cannot be added or deleted).

  `DCTAM_STS_WRITE`: The TAM table is update type (records can be added or deleted).

  `DCTAM_STS_RECLCK`: The TAM table is update type (records can be added and deleted without locking the table).

- `st_lod_type`

  The loading opportunity of the TAM table is returned as follows:

  `DCTAM_STS_START`: The TAM table is loaded when the TAM service is started.

  `DCTAM_STS_LIB`: The TAM table is loaded when the TAM table is opened by the function `dc_tam_open()`.

  `DCTAM_STS_CMD`: The TAM table is loaded when the `tamload` command is executed.

- `reserve1`

  Reserved area

- `st_rec_len`

  The record length of the TAM table is returned.

- `st_key_len`

  The key length of the TAM table is returned.

- `st_key_pos`

  The key start position in the TAM table data is returned.

- `st_tbl_sec`

  The security attribute of the TAM table specified in the TAM service definition is returned as follows:

DCTAM_STS_NOSEC: Security is not specified.

DCTAM_STS_SEC: Security is specified.

- reserve2

  Reserved area

Return values

Return value	Return value (numeric)	Explanation
DC_OK	0	Information was acquired from the TAM table normally.
DCTAMER_PARAM_TBL	-1701	The value specified for tblname is invalid.
DCTAMER_PARAM_FLG	-1708	The value specified for flags is invalid.
DCTAMER_NOTTAM	-1709	The name specified for tblname is not a TAM file name.
DCTAMER_UNDEF	-1710	The TAM table has not been defined.
DCTAMER_TAMEND	-1720	The TAM service is being terminated.
DCTAMER_PROTO	-1721	The sequence of accessing the TAM table is invalid.
		The resource manager registration of the object file for transactions control having a linkage with the UAP is invalid. Alternatively, there is no linkage between the object file for control of transactions and the UAP.
		atomic_update=N (nontransaction attribute) is specified in the user service definition of the UAP which called the function.
DCTAMER_TBLVR	-1760	The version of the TAM library linked to the UAP does not allow the UAP to operate with the current TAM table.
DCTAMER_TAMVR	-1762	The version of the TAM library linked to the UAP does not allow the UAP to operate with the current TAM service.
DCTAMER_OPENNUM	-1766	The number of open character special files exceeds the specified limit.
DCTAMER_ACCESSS	-1767	The access permission for special files has not been granted.
DCTAMER_MEMORY	-1769	The memory became insufficient.

Return value	Return value (numeric)	Explanation
DCTAMER_IO	-1770	An input/output error occurred.
DCTAMER_NO_ACL	-1772	The TAM table from which information is acquired is protected with the security facility. There is no ACL for the corresponding TAM table.
DCTAMER_ACCESS	-1773	The TAM table from which information is acquired is protected with the security facility. The UAP that called the function dc_tam_status() has no access permission.

dc_tam_write - Update/add a TAM table record

Format

- ANSI C, C++

```
#include <dctam.h>
int  dc_tam_write (DCLONG tblid, struct DC_TAMKEY *keyadr,
                   int keyno, char *datadr, int datsize,
                   DCLONG flags)
```

- K&R C

```
#include <dctam.h>
int  dc_tam_write (tblid, keyadr, keyno, datadr, datsize,
                   flags)
DCLONG    tblid;
struct DC_TAMKEY *keyadr;
int       keyno;
char      *datadr;
int       datsize;
DCLONG    flags;
```

Description

The function `dc_tam_write()` updates/adds a record indicated with a key value in/to a TAM table.

If a TAM table is open under lock in records, the following lock is enabled:

- When the access type is "update" (`DCTAM_WRITE` specified for `flags`):

  Lock in tables is enabled with lock for reference processing, and lock in records is enabled with lock for update processing.

  However, table lock is not enabled for tables whose access type is "reference" or "update without permission of addition or deletion" if "table nonlock mode" is specified as the "table lock mode for access" in the TAM service definition.

- When the access type is "update or addition" or "addition" (`DCTAM_WRTADD` or `DCTAM_ADD` specified for `flags`):

  Lock in tables is enabled with lock for update processing.

If the function `dc_tam_write()` returns with an error, all the resources specified within this function are released, and the status before this function was called is regained. However, if a TAM table which was acquired under lock for reference processing before this function was called is updated/added, lock for update processing is enabled. (Lock for reference processing is not regained.)

When a request is made to update/add multiple specified records, even if one of the

records causes an error, the processing of all the records specified in this function results in an error.

The key value storage location in the data to be updated/added and the key area length are as specified in the `tamcre` command used for creation of a TAM table file.

The data part has a key value if the key value is assigned to the data part (the `-s` option not specified in the `tamcre` command) when a TAM table file is created. Therefore, an error is returned if the key value specified in the function `dc_tam_write()` is not found in the data to be updated/added. The data part has no key value if no key value is assigned to the data part (the `-s` option specified in the `tamcre` command). In this case, no check is made on the contents of the data to be updated/added.

Arguments whose values are set in the UAP

- `tblid`

  Specify the table descriptor of the TAM table whose record is to be updated/added. The table descriptor is the value returned with the function `dc_tam_open()`.

- `keyadr`

  Specify the address of the structure having the address of the key value of the record to be updated/added. The structure format is as follows:

```
struct DC_TAMKEY {
                  char *keyname;
                 };
```

  - `keyname`

    Specify the address of the key value. The key value must be specified with the length of the key area of the record to be updated/added.

- `keyno`

  Specify the number of request records (number of structures specified for keyadr).

- `datadr`

  Specify the address of the data to be updated/added.

- `datsize`

  Specify the length of the data to be updated/added. The length of the data to be updated/added must be equal to or greater than (record length x number of request records).

- `flags`

  Specify in the format shown below the record access type and the lock release wait type when competition for a resource occurs:

```
{DCTAM_WRITE|DCTAM_WRTADD|DCTAM_ADD}[|{DCTAM_WAIT|DCTAM_NOWAIT}]
```

- Flag 1

  The specification of record access type cannot be omitted. More than one record access type cannot be specified at a time.

  DCTAM_WRITE: Update

  DCTAM_WRTADD: Update or addition

  DCTAM_ADD: Addition

- Flag 2

  More than one lock release wait type cannot be specified at a time. The default is DCTAM_NOWAIT.

  DCTAM_WAIT: The function waits for the resource to be released from lock.

  DCTAM_NOWAIT: The function does not wait for the resource to be released from lock, and returns with an error.

Return values

Return value	Return value (numeric)	Explanation
DC_OK	0	The TAM table record was updated/added normally.
DCTAMER_PARAM_TID	-1700	The table descriptor specified for tblid is invalid.
DCTAMER_PARAM_KEY	-1702	The key value specified for keyadr is invalid.
DCTAMER_PARAM_KNO	-1703	The value specified for keyno is invalid.
DCTAMER_PARAM_DTA	-1706	The value specified for datadr is invalid.
DCAMER_PARAM_DTS	-1707	The data length specified for datsize is too short.
DCTAMER_PARAM_FLG	-1708	The value specified for flags is invalid.
DCTAMER_NOTTAM	-1709	The table specified for tblid is not a TAM table.
DCTAMER_TAMEND	-1720	The TAM service is being terminated.
DCTAMER_PROTO	-1721	The sequence of accessing the TAM table is invalid.
		The resource manager registration of the object file for transaction control having a linkage with the UAP is invalid. Alternatively, there is no linkage between the object file for transaction control and the UAP.

Return value	Return value (numeric)	Explanation
		atomic_update=N (nontransaction attribute) is specified in the user service definition of the UAP which called the function.
DCTAMER_RMTBL	-1723	The TAM table was deleted.
DCTAMER_NOLOAD	-1724	The TAM table was not loaded.
DCTAMER_NOOPEN	-1726	The TAM table is not open.
DCTAMER_LOGHLD	-1727	The TAM table is in logical shutdown state.
DCTAMER_OBSHLD	-1728	The TAM table is in shutdown state due to an error.
DCTAMER_ACSATL	-1730	Execution is impossible in the access mode of the TAM table specified in the TAM service definition.
DCTAMER_NOREC	-1731	The specified record does not exist.
DCTAMER_EXKEY	-1735	The record cannot be added because the key value specified for keyadr exists in the TAM table.
DCTAMER_LOCK	-1736	A lock error occurred. If DCTAM_WAIT is specified for flags, the resource could not be acquired because a timeout occurred (the wait time specified in the lock service definition was exceeded).
DCTAMER_DLOCK	-1737	A deadlock occurred.
DCTAMER_TBLVR	-1760	The version of the TAM library linked to the UAP does not allow the UAP to operate with the current TAM table.
DCTAMER_FLSVR	-1761	The version of the TAM library linked to the UAP does not allow the UAP to operate with the current OpenTP1 file service.
DCTAMER_NOAREA	-1763	The TAM table has no free record.
DCTAMER_RECOBS	-1764	The record has been damaged.
DCTAMER_TRNNUM	-1765	The number of transactions exceeds the maximum number of transactions which can be managed by the TAM service.
DCTAMER_OPENNUM	-1766	The number of open character special files exceeds the specified limit.
DCTAMER_ACCESSS	-1767	The access permission for special files has not been granted.

Return value	Return value (numeric)	Explanation
DCTAMER_ACCESSF	-1768	The access permission for TAM table files has not been granted.
DCTAMER_MEMORY	-1769	The memory became insufficient.
DCTAMER_IO	-1770	An input/output error occurred.
DCTAMER_TMERR	-1771	A transaction service error occurred.
DCTAMER_ACCESS	-1773	A TAM file to be accessed is protected with the security facility. The UAP that called the function dc_tam_write() has no access permission.

Transaction control (dc_trn_~)

This section gives the syntax and other information of the following functions which are used for OpenTP1-specific transaction control:

- `dc_trn_begin` - Start a transaction
- `dc_trn_chained_commit` - Enable commitment in chained mode
- `dc_trn_chained_rollback` - Enable rollback in chained mode
- `dc_trn_info` - Report the information about the current transaction
- `dc_trn_unchained_commit` - Enable commitment in unchained mode
- `dc_trn_unchained_rollback` - Enable rollback in unchained mode

The functions for transaction control (`dc_trn_~`) can be used in UAPs of both TP1/Server Base and TP1/LiNK.

dc_trn_begin - Start a transaction

Format

■ ANSI C, C++

```
#include <dctrn.h>
int  dc_trn_begin (void)
```

■ K&R C

```
#include <dctrn.h>
int dc_trn_begin()
```

Description

The function `dc_trn_begin()` starts a global transaction from the process that calls this function. The process that called the function `dc_trn_begin()` becomes the root transaction branch of the global transaction.

For the UAP which calls the function `dc_trn_begin()`, specify the transaction attribute at execution environment setup.

Once the function `dc_trn_begin()` is called in a global transaction, the function `dc_trn_begin()` cannot be recalled from any transaction branch of the global transaction. If the function `dc_trn_begin()` is called more than once in a global transaction, an error is returned.

Return values

Return value	Return value (numeric)	Explanation
`DC_OK`	`0`	Normal termination. A global transaction was generated, and the process that called the function `dc_trn_begin()` is in the range of the global transaction.
`DCTRNER_PROTO`	`-905`	The function `dc_trn_begin()` was called from an invalid context (e.g., already in the transaction). Alternatively, the transaction could not be started because the execution environment was in non-journal operation mode.
`DCTRNER_RM`	`-906`	A resource manager (RM) error occurred. A transaction could not be started.

Return value	Return value (numeric)	Explanation
DCTRNER_TM	-907	A transaction could not be started because a transaction service error occurred. The value specified for the trn_tran_process_count operand in the transaction service definition may be insufficient. If this value is returned, reexecute processing. The reexecution is very likely to be successful.

Example

```
if(!dc_trn_info(NULL) &&dc_trn_begin() <0)
   fputs("cannot begin transaction\n", stderr);
```

dc_trn_chained_commit - Enable commitment in chained mode

Format

- ANSI C, C++

```
#include <dctrn.h>
int  dc_trn_chained_commit (void)
```

- K&R C

```
#include <dctrn.h>
int  dc_trn_chained_commit()
```

Description

The function `dc_trn_chained_commit()` acquires the synchronization point of a transaction. The normal termination of processing (commitment) is reported as the root transaction branch of the global transaction to the UAPs, transaction services, and resource managers of transaction branches which form the transaction.

When the function `dc_trn_chained_commit()` terminates normally, a new global transaction is started. The process that calls the function is in the range of this transaction. However, this does not mean the specification of a transaction mode for a UAP other than the UAP that called this function.

When a global transaction consists of multiple transaction branches (not only with the UAP that called the function), commitment processing is executed only when the processing results of each transaction branch are committed.

The function `dc_trn_chained_commit()` can be called only from the root transaction branch (the UAP that called the function `dc_trn_begin()`) of a global transaction. If the function `dc_trn_chained_commit()` is called from another UAP, `DCTRNER_PROTO` is returned.

Only the process that started the UAP executable file correctly linked according to the specification in this manual is permitted to call the function `dc_trn_chained_commit()`.

The function `dc_trn_chained_commit()` can terminate either normally or abnormally when synchronization point processing is completed. To have the function `dc_trn_chained_commit()` terminated normally, specify the transaction attribute at UAP execution environment setup.

Return values

Return value	Return value (numeric)	Explanation
DC_OK	0	Normal termination. Even after the function dc_trn_chained_commit() terminates, this process is under the transaction and it is in the range of the global transaction.
DCTRNER_ROLLBACK	-902	The current transaction was rolled back because it could not be committed. Even after this return value was returned, the process is still under the transaction and it is within the range of the global transaction.
DCTRNER_HEURISTIC	-903	The global transaction that called the function dc_trn_chained_commit() was determined heuristically. Consequently, a transaction branch was committed, and another transaction branch was rolled back. This value is returned if the results of heuristic decision do not match the results of the synchronization point of the global transaction. Refer to the message log file for the results of the synchronization point of the UAP, resource manager, or global transaction that caused this value to be returned. Even after this value is returned, this process is under the transaction and it is in the range of the global transaction.
DCTRNER_HAZARD	-904	A transaction branch of the global transaction was completed heuristically. However, the results of the synchronization point of the heuristically completed transaction branch are not known due to an error. Refer to the message log file for the results of the synchronization point of the UAP, resource manager, or global transaction that caused this value to be returned. Even after this value is returned, this process is under the transaction and it is in the range of the global transaction. This function returns DCTRNER_HAZARD even when you specify 00000001 for the trn_extend_function operand in the transaction service definition and the return value from the resource manager at one-phase commit is XAER_NOTA.
DCTRNER_PROTO	-905	The function dc_trn_chained_commit() was called from an invalid context (e.g., already not in the transaction). The transaction mode is not affected.

Return value	Return value (numeric)	Explanation
DCTRNER_NO_BEGIN	-924	Although the commitment processing terminated normally, the new transaction could not be started. After this value is returned, this process is not under the transaction.
DCTRNER_ROLLBACK_NO_BEGIN	-925	The transaction was rolled back because it could not be committed. The new transaction could not be started. After this value is returned, this process is not under the transaction.
DCTRNER_HEURISTIC_NO_BEGIN	-926	The global transaction that called the function dc_trn_chained_commit() was determined heuristically. Consequently, a transaction branch was committed, and another transaction branch was rolled back. This value is returned if the results of heuristic decision do not match the results of the synchronization point of the global transaction. See to the message log file for the results of the synchronization point of the UAP, resource manager, or global transaction that caused this value to be returned. The new transaction could not be started. After this value is returned, this process is not under the transaction.
DCTRNER_HAZARD_NO_BEGIN	-927	A transaction branch of the global transaction was completed heuristically. However, the results of the synchronization point of the heuristically completed transaction branch are not known due to an error. See to the message log file for the results of the synchronization point of the UAP, resource manager, or global transaction that caused this value to be returned. The new transaction could not be started. After this value is returned, this process is not under the transaction. This function returns DCTRNER_HAZARD_NO_BEGIN even when you specify 00000001 for the trn_extend_function operand in the transaction service definition and the return value from the resource manager at one-phase commit is XAER_NOTA.

Example

```
if(dc_trn_info(NULL)&&dc_trn_chained_commit() <0)
 fputs("cannot commit transaction\n", stderr);
```

dc_trn_chained_rollback - Enable rollback in chained mode

Format

- ANSI C, C++

```
#include <dctrn.h>
int  dc_trn_chained_rollback (void)
```

- K&R C

```
#include <dctrn.h>
int  dc_trn_chained_rollback()
```

Description

The function `dc_trn_chained_rollback()` rolls back a transaction. A transaction is started immediately after the function `dc_trn_chained_rollback()` is called.

To call the function `dc_trn_chained_rollback()`, rollback processing is reported from the root transaction branch of the global transaction to the UAPs, transaction services, and resource managers of transaction branches which form the transaction.

When the function `dc_trn_chained_rollback()` terminates normally, the process that called the function returns after rollback processing. Then, a new global transaction is started. The process that calls the function is in the range of this transaction. However, this does not mean the specification of a transaction mode for a UAP other than the UAP that called this function.

The function `dc_trn_chained_rollback()` can be called only from the root transaction branch (the UAP that called the function `dc_trn_begin()`) of a global transaction. If the function `dc_trn_chained_rollback()` is called from another UAP, `DCTRNER_PROTO` is returned.

Only the process that started the UAP which is created correctly according to the specification in this manual is permitted to call the function `dc_trn_chained_rollback()`.

The function `dc_trn_chained_rollback()` can terminate either normally or abnormally when synchronization point processing is completed. To have the service which calls the function `dc_trn_chained_rollback()` terminated normally, specify the transaction attribute at UAP execution environment setup.

Return values

Return value	Return value (numeric)	Explanation
DC_OK	0	Normal termination. Even after the function dc_trn_chained_rollback() terminates, this process is under the transaction and it is in the range of the global transaction.
DCTRNER_HEURISTIC	-903	The global transaction that called the function dc_trn_chained_rollback() was determined heuristically. Consequently, a transaction branch was committed, and another transaction branch was rolled back. This value is returned if the results of heuristic decision do not match the results of the synchronization point of the global transaction. Refer to the message log file for the results of the synchronization point of the UAP, resource manager, or global transaction that caused this value to be returned. Even after this value is returned, this process is under the transaction and it is in the range of the global transaction.
DCTRNER_HAZARD	-904	A transaction branch of the global transaction was completed heuristically. However, the results of the synchronization point of the heuristically completed transaction branch are not known due to an error. Refer to the message log file for the results of the synchronization point of the UAP, resource manager, or global transaction that caused this value to be returned. Even after this value is returned, this process is under the transaction and it is in the range of the global transaction.
DCTRNER_PROTO	-905	The function dc_trn_chained_rollback() was called from an invalid context (e.g., already not in the transaction). The transaction mode is not affected.
DCTRNER_NO_BEGIN	-924	Although the rollback processing terminated normally, the new transaction could not be started. After this value is returned, this process is not under the transaction.

Return value	Return value (numeric)	Explanation
DCTRNER_HEURISTIC_NO_BEGIN	-926	The global transaction that called the function dc_trn_chained_rollback() was determined heuristically. Consequently, a transaction branch was committed, and another transaction branch was rolled back. This value is returned if the results of heuristic decision do not match the results of the synchronization point of the global transaction. See to the message log file for the results of the synchronization point of the UAP, resource manager, or global transaction that caused this value to be returned. The new transaction could not be started. After this value is returned, this process is not under the transaction.
DCTRNER_HAZARD_NO_BEGIN	-927	A transaction branch of the global transaction was completed heuristically. However, the results of the synchronization point of the heuristically completed transaction branch are not known due to an error. See to the message log file for the results of the synchronization point of the UAP, resource manager, or global transaction that caused this value to be returned. The new transaction could not be started. After this value is returned, this process is not under the transaction.

Example

```
if (dc_trn_info (NULL) && dc_trn_chained_rollback () <0)
  fputs("cannot rollback transaction\n", stderr);
```

Note

This API does not obtain a UAP trace.

dc_trn_info - Report the information about the current transaction

Format

■ ANSI C, C++

```
#include <dctrn.h>
int  dc_trn_into (char *flags)
```

■ K&R C

```
#include <dctrn.h>
int  dc_trn_info (flags)
char      *flags;
```

Description

The function `dc_trn_info()` returns information which indicates whether the UAP that called the function `dc_trn_info()` is operating as the current transaction.

Only the process that started the UAP which is created correctly according to the specification in this manual is permitted to call the function `dc_trn_info()`. To have the service which calls the function `dc_trn_info()` terminated normally, specify the transaction attribute at UAP execution environment setup.

Argument whose value is set in the UAP

■ `flags`

Specify a `NULL`.

Return values

Return value	Explanation
`1`	The process that called the function `dc_trn_info()` is operating as a transaction.
`0`	The process that called the function `dc_trn_info()` is not operating as a transaction.

Example

```
if(!dc_trn_info(NULL)&&dc_trn_begin() <0)
  fputs("cannot begin transaction\n", stderr);
```

Note

This API does not obtain a UAP trace.

dc_trn_unchained_commit - Enable commitment in unchained mode

Format

■ ANSI C, C++

```
#include <dctrn.h>
int  dc_trn_unchained_commit (void)
```

■ K&R C

```
#include <dctrn.h>
int  dc_trn_unchained_commit()
```

Description

The function `dc_trn_unchained_commit()` posts the normal termination of a global transaction (commitment) to the UAPs, transaction services, and resource managers of transaction branches which form the transaction. After the function `dc_trn_unchained_commit()` terminates normally, a new global transaction is not started.

When a global transaction consists of multiple transaction branches (not only with the UAP that called the function), commitment processing is executed only when the processing results of each transaction branch is committed.

The function `dc_trn_unchained_commit()` can be called only from the root transaction branch (the UAP that started the transaction). If the function is called from any other transaction, it returns with an error, giving the return value `DCTRNER_PROTO`.

Only the process that started the UAP which is created correctly according to the specification in this manual is permitted to call the function `dc_trn_unchained_commit()`.

The function `dc_trn_unchained_commit()` can terminate either normally or abnormally when synchronization point processing is completed. To have the service which calls the function `dc_trn_unchained_commit()` terminated normally, specify the transaction attribute at UAP execution environment setup.

Return values

Return value	Return value (numeric)	Explanation
`DC_OK`	0	Normal termination. This process is not under the transaction and it is not in the range of the global transaction.

Return value	Return value (numeric)	Explanation
DCTRNER_ROLLBACK	-902	The current transaction was rolled back because it could not be committed. This process is not in the range of the global transaction.
DCTRNER_HEURISTIC	-903	The global transaction that called the function dc_trn_unchained_commit() was determined heuristically. Consequently, a transaction branch was committed, and another transaction branch was rolled back. This value is returned if the results of heuristic decision do not match the results of the synchronization point of the global transaction. Refer to the message log file for the results of the synchronization point of the UAP, resource manager, or global transaction that caused this value to be returned. After this value is returned, this process is not under the transaction and it is not in the range of the global transaction.
DCTRNER_HAZARD	-904	A transaction branch of the global transaction was completed heuristically. However, the results of the synchronization point of the heuristically completed transaction branch are not known due to an error. Refer to the message log file for the results of the synchronization point of the UAP, resource manager, or global transaction that caused this value to be returned. After this value is returned, this process is not under the transaction and it is not in the range of the global transaction. This function returns DCTRNER_HAZARD even when you specify 00000001 for the trn_extend_function operand in the transaction service definition and the return value from the resource manager at one-phase commit is XAER_NOTA.
DCTRNER_PROTO	-905	The function dc_trn_unchained_commit() was called from an invalid context (e.g., already not in the transaction). The transaction mode is not affected.

Example

```
if(dc_trn_info(NULL) &&dc_trn_unchained_commit() <0)
  fputs("cannot commit transaction\n", stderr);
```

dc_trn_unchained_rollback - Enable rollback in unchained mode

Format

- ANSI C, C++

```
#include <dctrn.h>
int  dc_trn_unchained_rollback (void)
```

- K&R C

```
#include <dctrn.h>
int  dc_trn_unchained_rollback()
```

Description

The function `dc_trn_unchained_rollback()` rolls back a transaction. If a transaction is rolled back in unchained mode, the transaction does not start contiguously.

Calling the function `dc_trn_unchained_rollback()` notifies a transaction branch, transaction service, and resource manager of a rollback.

The function `dc_trn_unchained_rollback()` can be called from any transaction branch of a global transaction. If the function `dc_trn_unchained_rollback()` is called from the root transaction branch, a new transaction does not start after the function `dc_trn_unchained_rollback()` returns normally.

If the function `dc_trn_unchained_rollback()` is called from a transaction branch other than the root transaction branch, the function `dc_trn_unchained_rollback()` puts the transaction branch into `rollback_only` state. In this case, the transaction branch that called the function `dc_trn_unchained_rollback()` is in the range of the transaction until synchronization point processing of the root transaction branch is completed.

Only the process that started the UAP which is created correctly according to the specification in this manual is permitted to call the function `dc_trn_unchained_rollback()`. To have the service which calls the function `dc_trn_unchained_rollback()` terminated normally, specify the transaction attribute at UAP execution environment setup.

Return values

Return value	Return value (numeric)	Explanation
DC_OK	0	Normal termination. If the function dc_trn_unchained_rollback() is called from the root transaction branch, this process is not under the transaction and it is not in the range of the global transaction. If the function dc_trn_unchained_rollback() is called from a transaction branch other than the root transaction branch, this process is put into rollback_only state.
DCTRNER_HEURISTIC	-903	The global transaction that called the function dc_trn_unchained_rollback() was determined heuristically. Consequently, a transaction branch was committed, and another transaction branch was rolled back. This value is returned if the results of heuristic decision do not match the results of the synchronization point of the global transaction. Refer to the message log file for the results of the synchronization point of the UAP, resource manager, or global transaction that caused this value to be returned. After this value is returned, this process is not under the transaction and it is not in the range of the global transaction.
DCTRNER_HAZARD	-904	A transaction branch of the global transaction was completed heuristically. However, the results of the synchronization point of the heuristically completed transaction branch are not known due to an error. Refer to the message log file for the results of the synchronization point of the UAP, resource manager, or global transaction that caused this value to be returned. After this value is returned, this process is not under the transaction and it is not in the range of the global transaction.
DCTRNER_PROTO	-905	The function dc_trn_unchained_rollback() was called from an invalid context (e.g., already not in the transaction). The transaction mode is not affected.

Example

```
if (dc_trn_info (NULL) && dc_trn_unchained_rollback () <0)
  fputs ("cannot rollback transaction\n", stderr);
```

Online tester management (dc_uto_~)

This section gives the functions used to maintain the status of the online tester from a user server while the online tester (TP1/Online Tester) is used under the OpenTP1. The syntax of the following function is explained:

- `dc_uto_test_status()` - Report the test status of a user server

  The function for online tester management (`dc_uto_~`) can be used only in UAPs of TP1/Server Base. They cannot be used in UAPs of TP1/LiNK.

dc_uto_test_status - Report the test status of a user server

Format

- ANSI C, C++

```
#include <dcuto.h>
int  dc_uto_test_status (struct DC_UTOSTAT *test_stat,
                         DCLONG flags)
```

- K&R C

```
#include <dcuto.h>
int  dc_uto_test_status (test_stat, flags)
struct  DC_UTOSTAT  *test_stat;
DCLONG     flags;
```

Description

The function `dc_uto_test_status()` returns the status of testing a user server that called this function. The test status is stored in an argument after this function returns normally. If this function returns with an error, information on the test status stored in the argument is undefined.

Arguments whose values are set in the UAP

- `test_stat`

  Specify the address of the structure indicating the status of testing a user server.

- `flags`

  Specify `DCNOFLAGS`.

Arguments whose values are returned from the OpenTP1

- `test_stat`

  Information on the status of testing a user server is returned with a structure. The structure formats are as follows:

```
struct  DC_UTOSTAT {
                      char  testID[5];
                      char  mode;
                      char  gbl_tran;
                      char  type;
                      char  svr_tran;
                      char  comd;
                      char  _res[22];
};
```

- testID

  The test user ID (the value set for the environment variable DCUTOKEY) is returned.

- mode

  Whether the user server is operating in the test mode is returned.

  DCUTO_TEST: Operating in the test mode.

  DCUTO_NOTEST: Not operating in the test mode.

- gbl_tran

  The processing status of the global transaction is returned.

  DCUTO_TRN_COMMIT: Commits in the synchronization point processing.

  DCUTO_TRN_ROLLBACK: Rolls back in the synchronization point processing.

  DCUTO_TRN_NOTRN: Non-transaction status

  NULL (null character): Non-test mode. MHP linked with an MCF library.

- type

  The test type specified in the test_mode operand of the user service definition is returned.

  DCUTO_TEST_MODE_TARGET: UAP dedicated to the test (target)

  DCUTO_TEST_MODE_USABLE: Usable UAP (usable)

  DCUTO_TEST_MODE_SIMMHP: Simulate MHP (simmhp)

  DCUTO_TEST_MODE_NO： UAP ineligible for the test (no)

- svr_tran

  Method of handling the transaction at the synchronization point specified in the test_transaction_commit operand of the user service definition is returned.

  DCUTO_TRN_COMMIT: Commits (Y) at the synchronization point.

  DCUTO_TRN_ROLLBACK: Rolls back (N) at the synchronization point.

  NULL (null character): Non-test mode. MHP linked with an MCF library.

- comd

  Method of handling the command execution result specified in the test_adm_call_commit operand of the user service definition is returned.

  DCUTO_COMMAND_DO: Executes the command (do).

  DCUTO_COMMAND_SKIP: Sets an assumption value as an execution result (skip).

DCUTO_COMMAND_FILE: Uses data in the operation command result data file (file).

NULL (null character): Non-test mode. MHP linked with an MCF library.

Return values

Return value	Return value (numeric)	Explanation
DC_OK	0	Normal termination. The test status is returned in the area indicated by the structure DC_UTOSTAT.
DCUTOER_PROTO	-2701	The function dc_rpc_open() is not called.
DCUTOER_TRAN	-2734	The UAP is linked with an OpenTP1 library which is inoperable with the current transaction service.
DCUTOER_PARAM_FLAGS	-2757	The value specified for flags is invalid.
DCUTOER_PARAM_ADDS	-2758	The value specified for test_stat is invalid.

Note

When the function dc_uto_test_status() is called from an MHP, the following values are returned to the structure DC_UTOSTAT:

- testID: Test user ID
- mode: Current service mode
- gbl_tran: Null character
- type: DCUTO_TEST_MODE_NO
- svr_tran: Null character
- comd: Null character

Chapter

3. Syntax of OpenTP1 Library Functions (Message Log Reporting)

A message log can be reported so that the status of OpenTP1 is reported to products other than OpenTP1. This chapter explains the syntax of the OpenTP1 library functions for receiving message logs to obtain the status of OpenTP1.

This chapter contains the following section:

Message log reporting (dc_log_~)

Message log reporting (dc_log_~)

This section explains the syntax of the OpenTP1 library functions for receiving message logs to obtain the status of OpenTP1. The functions for reporting message logs are as follows:

- `dc_log_notify_close` - Terminate message log reception
- `dc_log_notify_open` - Start message log reception
- `dc_log_notify_receive` - Receive message logs
- `dc_log_notify_send` - Send user-kept message logs

The function (`dc_log_~`) for reporting message logs can be used only for TP1/Server Base. For TP1/LiNK, the function cannot be used.

Only the application programs created for reception can receive message logs. OpenTP1 UAPs (SUP, SPP, and MHP) cannot receive message logs.

Notes on receiving message logs

Note the following when receiving message logs:

1. The functions `dc_log_notify_open()`, `dc_log_notify_receive()`, and `dc_log_notify_close()` cannot be executed in the interrupt routine.
2. Some message logs cannot be received, depending on when the function `dc_log_notify_receive()` is called. The following message logs cannot be received:
   - Message logs output by OpenTP1 while the application program is stopped, before the program calls the function `dc_log_notify_open()`, or after it calls the function `dc_log_notify_close()`.
   - Message logs reported by OpenTP1 after the save area becomes full if the function `dc_log_notify_receive()` is not called.

dc_log_notify_close - Terminate message log reception

Format

- ANSI C, C++

```
#include <dclog.h>
DCLONG    dc_log_notify_close (DCLONG flags)
```

- K&R C

```
#include <dclog.h>
DCLONG    dc_log_notify_close (flags)
DCLONG     flags;
```

Description

The function `dc_log_notify_close()` terminates reception of the message logs reported by OpenTP1. Calling the function `dc_log_notify_open()` again restarts message log reception.

Argument whose value is set in the UAP

- `flags`

  Specify `DCNOFLAGS`.

Return values

Return value	Return value (numeric)	Meaning
`DC_OK`	`0`	Normal termination.
`DCLOGER_PARAM_ARGS`	`-1900`	An incorrect value is specified as the argument.
`DCLOGER_PROTO`	`-1999`	The function `dc_log_notify_open()` is not called.

dc_log_notify_open - Start message log reception

Format

- ANSI C, C++

```
#include <dclog.h>
DCLONG    dc_log_notify_open (DCLONG id, DCLONG flags)
```

- K&R C

```
#include <dclog.h>
DCLONG  dc_log_notify_open (id, flags)
DCLONG     id;
DCLONG     flags;
```

Description

The function `dc_log_notify_open()` starts reception of the message logs reported by OpenTP1.

Arguments whose values are set in the UAP

- `id`

  Specify 0.

- `flags`

  `DCNOFLAGS`

  Specify this value if you do not want to determine whether the use of the message log report facility operand is specified in the log service definition.

  `DCLOG_CHKRTN`

  Specify this value if you want to determine whether the use of the message log report facility operand is specified in the log service definition. If the use of the operand is not specified, `DCLOGER_PROTO` will return.

Return values

Return value	Return value (numeric)	Meaning
`DC_OK`	`0`	Normal termination.
`DCLOGER_PARAM_ARGS`	`-1900`	An incorrect value is specified as the argument.

Return value	Return value (numeric)	Meaning
DCLOGER_PROTO	-1999	The function dc_log_notify_open() has already been called. If DCLOG_CHKRTN is assigned to flags, the log service definition is specified so that the message log report facility will not be used.
DCLOGER_DEFFILE	-1904	The system definition is invalid.
DCLOGER_MEMORY	-1902	The memory became insufficient.
DCLOGER_COMM	-1901	Initialization of the communication path failed.

dc_log_notify_receive - Receive message logs

Format

- ANSI C, C++

```
#include <dclog.h>
DCLONG  dc_log_notify_receive (char *msg, DCLONG msglen,
                               DCLONG timeout, DCLONG flags)
```

- K&R C

```
#include <dclog.h>
DCLONG  dc_log_notify_receive (msg, msglen, timeout, flags)
char       *msg;
DCLONG     msglen;
DCLONG     timeout;
DCLONG     flags;
```

Description

The function `dc_log_notify_receive()` receives the message logs reported by OpenTP1. Calling the function `dc_log_notify_receive()` once retrieves one message log.

Arguments whose values are set in the UAP

- `msg`

  Specify the area to contain a receive message log. Here, specify length greater than or equal to that specified in `DCLOG_NOTIFY_MSG_LEN`.

- `msglen`

  Specify the length of the area specified in `msg`.

- `timeout`

  Specify the time (in seconds) during which the function `dc_log_notify_receive()` waits if no message log arrives. The number of seconds must be from -1 to 65,535. If 0 is specified, the function returns without waiting for message logs. If -1 is specified, the function waits until a message log arrives.

- `flags`

  Specify `DCNOFLAGS`.

Return values

Return value	Return value (numeric)	Meaning
Integer of 0 or larger		A message log was stored normally in the area specified in `msg`. An integer of 0 or larger indicates the length of the received message log.
`DCLOGER_PARAM_ARGS`	`-1900`	An incorrect value is specified as the argument.
`DCLOGER_PROTO`	`-1999`	The function `dc_log_notify_open()` is not called.
`DCLOGER_TIMEOUT`	`-1907`	Though the number of seconds specified in `timeout` is exceeded, no message log is reported.
`DCLOGER_COMM`	`-1901`	Initialization of the communication path failed.

dc_log_notify_send - Send user-kept message logs

Format

- ANSI C, C++

```
#include <dclog.h>
DCLONG  dc_log_notify_send (char *msg, DCLONG msglen,
                            DCLONG flags)
```

- K&R C

```
#include <dclog.h>
DCLONG  dc_log_notify_send (msg, msglen, flags)
char       *msg;
DCLONG     msglen;
DCLONG     flags;
```

Description

The function dc_log_notify_send() sends a message log kept optionally by the user to the application programs waiting for message logs from OpenTP1. The function is used for requesting termination of the application programs waiting for message logs.

Arguments whose values are set in the UAP

- msg

  Specify the area containing a send message log.

- msglen

  Specify the length of the area specified in msg. Here, specify length shorter than or equal to that specified in DCLOG_NOTIFY_MSG_LEN.

- flags

  Specify DCNOFLAGS.

Return values

Return value	Return value (numeric)	Meaning
DC_OK	0	Normal termination.
DCLOGER_PROTO	-1999	The function dc_log_notify_open() has already been called. Therefore, the applicable application program cannot call the function dc_log_notify_send().

Return value	Return value (numeric)	Meaning
DCLOGER_PARAM_ARGS	-1900	An incorrect value is specified as the argument.
DCLOGER_COMM	-1901	Initialization of the communication path failed.

Chapter

4. X/Open-compliant Application Programming Interface

This chapter explains library functions conforming to the application programming interface based on X/Open.

This chapter contains the following sections:

X/Open-compliant function

Table 4-1 shows the correspondence between the X/Open-compliant functions (XATMI-interfaced or TX-interfaced) and their facilities, and Table 4-2 shows the relationship between these functions and OpenTP1 UAPs.

Table 4-1: Relationship between X/Open-compliant functions and facilities

Category	X/Open-compliant function - name and facility	
XATMI interface	`tpacall()`	Send a service request.
	`tpadvertise()`	Advertise a service name.
	`tpalloc()`	Allocate a typed buffer.
	`tpcall()`	Send a service request and synchronously awaits its reply.
	`tpcancel()`	Cancel a call descriptor for an outstanding reply.
	`tpconnect()`	Establish a conversational service connection.
	`tpdiscon()`	Terminate a conversational service connection abortively.
	`tpfree()`	Free a typed buffer.
	`tpgetrply()`	Get a reply from a previous service request.
	`tprealloc()`	Change the size of a typed buffer.
	`tprecv()`	Receive a message in a conversational connection.
	`tpreturn()`	Return from a service routine.
	`tpsend()`	Send a message in a conversational connection.
	`tpservice()`	Template for service routines.
	`tptypes()`	Determine information about a typed buffer.
	`tpunadvertise()`	Unadvertise a service name.
TX interface	`tx_begin()`	Begin a global transaction.
	`tx_close()`	Close a set of resource managers.

Category	X/Open-compliant function - name and facility	
	tx_commit()	Commit a global transaction.
	tx_info()	Return global transaction information.
	tx_open()	Open a set of resource managers.
	tx_rollback()	Roll back a global transaction.
	tx_set_commit_return()	Set commit_return characteristic.
	tx_set_transaction_control()	Set transaction_control characteristic.
	tx_set_transaction_timeout()	Set transaction_timeout characteristic.

Table 4-2: Relationship between X/Open-compliant functions and OpenTP1 UAPs

X/Open-compliant function	SUP		SPP			MHP		Off-line
	Out	In	Out	Transaction range		Out	In	
				Rt	N-Rt			
tpacall	Y	Y	Y	Y	Y	N	N	N
tpadvertise	N	N	Y#1	Y#1	Y#1	N	N	N
tpalloc	Y	Y	Y	Y	Y	N	N	N
tpcall	Y	Y	Y	Y	Y	N	N	N
tpcancel	Y	Y	Y	Y	Y	N	N	N
tpconnect	Y	Y	Y	Y	Y	N	N	N
tpdiscon	Y	Y	Y	Y	Y	N	N	N
tpgetrply	Y	Y	Y	Y	Y	N	N	N
tpfree	Y	Y	Y	Y	Y	N	N	N
tprecv	Y	Y	Y	Y	Y	N	N	N
tprealloc	Y	Y	Y	Y	Y	N	N	N
tpreturn	N	N	Y#2	Y#2	Y#2	N	N	N
tpsend	Y	Y	Y	Y	Y	N	N	N

X/Open-compliant function	SUP		SPP			MHP		Off-line
	Out	In	Out	Transaction range		Out	In	
				Rt	N-Rt			
tpservice[#3]	N[#3]	N[#3]	N[#3]	N[#3]	N[#3]	N	N	N
tptypes	Y	Y	Y	Y	Y	N	N	N
tpunadvertise	N	N	Y[#1]	Y[#1]	Y[#1]	N	N	N
tx_begin[#4]	Y	N	Y	N	N	Y	N	N
tx_close	Y	N	Y	N	N	N	N	N
tx_commit with TX_CHAINED specified[#4]	N	Y	Y	N	N	N	N	N
tx_commit with TX_UNCHAINED specified[4]	N	Y	Y	N	N	N	N	N
tx_info	Y	Y	Y	Y	Y	N	N	N
tx_open	Y	N	Y	N	N	N	N	N
tx_rollback with TX_CHAINED specified[#4]	N	Y	N	Y	N	N	N	N
tx_rollback with TX_UNCHAINED specified[#4]	N	Y	N	Y	N	N	N	N
tx_set_commit_return[#4]	Y	Y	Y	Y	Y	N	N	N
tx_set_transaction_control[#4]	Y	Y	Y	Y	Y	N	N	N
tx_set_transaction_timeout[#4]	Y	Y	Y	Y	Y	N	N	N

Legend:

Out: Outside transaction range

In: Inside transaction range (root)

Rt: Root

N-Rt: Non-root

Off-line: UAP that handles offline work

Y:The function can be used with UAPs.

N:The function cannot be used with UAPs.

The *Outside transaction range* for MHP means the range of MHPs with the nontransaction attribute or the main function of MHPs.

#1: Functions marked [#1] can be called only within service functions.

#2: Functions marked [#2] are used only to make XATMI-interfaced service functions return.

#3: `tpservice` is the entity of the service function.

#4: For the UAP which issues a function marked [#4], specify `atomic_update=Y` in the user service definition.

XATMI-interfaced application programming interface (tp~)

This section explains the syntax of the API functions which implement the XATMI interface. The text in this section is quoted from *5. C Reference Manual Pages* which is the syntax reference section of the *X/Open CAE Specification Distributed TP: The XATMI Specification* published by X/Open Company Limited.

Additional notes on using these functions from UAPs used with the OpenTP1 are enclosed in symbols << >>.

The syntax of the following functions is explained below:

- `tpacall` - Send a service request
- `tpadvertise` - Advertise a service name
- `tpalloc` - Allocate a typed buffer
- `tpcall` - Send a service request and synchronously await its reply
- `tpcancel` - Cancel a call descriptor for an outstanding reply
- `tpconnect` - Establish a conversational service connection
- `tpdiscon` - Terminate a conversational service connection abortively
- `tpfree` - Free a typed buffer
- `tpgetrply` - Get a reply from a previous service request
- `tprealloc` - Change the size of a typed buffer
- `tprecv` - Receive a message in a conversational connection
- `tpreturn` - Return from a service routine
- `tpsend` - Send a message in a conversational connection
- `tpservice` - Template for service routines
- `tptypes` - Determine information about a typed buffer
- `tpunadvertise` - Unadvertise a service name

XATMI interface functions (`tp~`) can be used only for TP1/Server Base. For TP1/LiNK, XATMI interface functions cannot be used.

tpacall - Send a service request

Format

- ANSI C, C++

```
#include <xatmi.h>
int tpacall (char *svc, char *data, long len,
             long flags)
```

- K&R C

```
#include <xatmi.h>
int tpacall (svc, data, len, flags)
char    *svc;
char    *data;
long    len;
long    flags;
```

Description

The function `tpacall()` sends a request message to the service named by `svc`. If `data` is non-NULL, it must point to a buffer previously allocated by `tpalloc()` and len should specify the amount of data in the buffer that should be sent. Note that if data points to a buffer of a type that does not require a length to be specified, `len` is ignored (and may be 0). If `data` is NULL, `len` is ignored and a request is sent with no data portion. The type and sub-type of `data` must match one of the types and sub-types recognized by `svc`. Note that for each request sent while in transaction mode, a corresponding reply must ultimately be received.

<<Arguments>>

- <<`svc`

  Specify the name of the service to be requested.>>

- <<`data`

  Specify the pointer to the send data storage area.>>

- <<`len`

  Specify the length of the send data. >>

- <<`flags`

  The valid flags are as follows:>>

  `TPNOTRAN`

  If the caller is in transaction mode and this flag is set, when `svc` is invoked, it is

not performed on behalf of the caller's transaction. If svc does not support transactions, this flag must be set when the caller is in transaction mode. A caller in transaction mode that sets this flag is still subject to the transaction timeout (and no other). If a service fails that was invoked with this flag, the caller's transaction is not affected.

TPNOREPLY

This setting informs tpacall() that a reply is not expected. When TPNOREPLY is set, the function returns 0 on success, where 0 is an invalid descriptor. When the caller is in transaction mode, this setting cannot be used unless TPNOTRAN is also set.

TPNOBLOCK

The request is not sent if a blocking condition exists (for example, the internal buffers into which the message is transferred are full). When TPNOBLOCK is not specified and a blocking condition exists, the caller blocks until the condition subsides or a timeout occurs (either transaction or blocking timeout).

TPNOTIME

This flag signifies the caller is willing to block indefinitely and wants to be immune to blocking timeouts. Transaction timeouts may still occur.

TPSIGRSTRT

If a signal interrupts any underlying system calls, the interrupted system call is reissued.

Return value

Upon successful completion, tpacall() returns a descriptor that can be used to receive the reply of the request sent. Otherwise it returns -1 and sets tperrno to indicate the error condition.

- Errors

Under the following conditions, tpacall() fails and sets tperrno to one of the values below. Unless otherwise noted, failure does not affect the caller's transaction, if one exists.

Return value	Return value (numeric)	Explanation
TPEINVAL	4	Invalid arguments were given (for example, svc is NULL, data does not point to a space allocated with tpalloc(), or the value of flags is invalid).
TPENOENT	6	Cannot establish a connection because the service specified in svc does not exist.

Return value	Return value (numeric)	Explanation
TPEITYPE	17	type and subtype for data are not in a format that can be used for svc.
TPELIMIT	5	The caller's request was not sent because the maximum number of outstanding asynchronous requests has been reached.
TPETRAN	14	TPNOTRAN was not set, even though transaction processing could not be performed for svc.
TPETIME	13	A timeout occurred. If the caller is in transaction mode, a transaction time-out occurred and the transaction is marked rollback_only; otherwise, a blocking time-out occurred and neither TPNOBLOCK nor TPNOTIME were specified. If a transaction time-out occurred, any attempts to send new requests or receive outstanding replies fail with TPETIME until the transaction has been rolled back.
TPEBLOCK	3	When tpacall() for which TPNOBLOCK was specified was called, the blocking status existed.
TPEGOTSIG	15	A signal was received, but TPSIGRSTRT was not set.
TPEPROTO	9	tpacall() was called in an improper context.
TPESYSTEM	12	A communication resource manager system error has occurred. The exact nature of the error is determined in a product-specific manner.
TPEOS	7	An operating system error has occurred. The exact nature of the error is determined in a product-specific manner.

See also

tpalloc(), tpcall(), tpcancel(), tpgetrply().

<<Notes on use with OpenTP1>>

1. <<The TPNOBLOCK flag is invalid under the relevant version of the OpenTP1. Therefore, the error code TPEBLOCK will not be returned to tperrno. The OpenTP1 is designed so that if communication is impossible because of blocking, TPESYSTEM is returned as when communication is impossible because of network failure.>>
2. <<The TPNOTIME flag is invalid under the relevant version of the OpenTP1.>>
3. <<The TPSIGRSTRT flag is invalid. Regardless of this flag, when a signal is received, the interrupted system call is reinvoked. TPEGOTSIG will never return.>>

4. <<Under the relevant version of the OpenTP1, TPEITYPE will not return. If data of a type unavailable with svc is passed, the function tpacall() normally returns, but TPESYSTEM will return when the function tpgetrply() is called. Therefore, the error condition is identified. If the calling program is in transaction mode, the rollback_only state comes into effect.>>
5. <<Under the OpenTP1, when a process encounters transaction timeout, it terminates abnormally. Therefore, TPETIME returns only when blocking timeout occurs.>>
6. <<Under the relevant version of the OpenTP1, data which requires rollback causes the return of TPESYSTEM unless otherwise specified by the X/Open. However, the rollback_only state may not come into effect even when TPESYSTEM returns.>>
7. <<Under the relevant version of the OpenTP1, TPELIMIT will not return.>>
8. <<For OSI TP communication using TP1/NET/OSI-TP-Extended, send data must not exceed the length specified in the length operand of the NET buffer group definition nettbuf (NET/Library common definition).>>
9. <<During OSI TP communication, the following conditions cause a TPESVCERR error when an attempt is made to issue the function tpcall() or tpgetrply(); during TCP/IP communication, they cause a TPENOENT or TPESYSTEM error when the same attempt is made:
   - The specified service does not exist at the request destination.
   - The typed buffer is not recognized by the server.
   - Service activation encounters an error.>>
10. <<If the number of system associations is insufficient during OSI TP communication, the function outputs a log message and returns with TPESYSTEM.>>
11. <<While OSI TP communication is in use, blocking time-out occurs even if TPNOTIME is specified. While TCP/IP communication is in use, blocking time-out occurs during non-transaction periods.>>
12. <<For OSI TP communication, the value assigned to the user service definition message_store_buflen must be equal to or greater than the size specified by nettbuf -g. For TCP/IP communication, the same rules as for the function dc_rpc_call() apply.>>
13. <<The behavior caused by XATMI errors encountered during OSI TP communication may be different from the behavior caused by errors encountered conventional TCP/IP communication.>>

tpadvertise - Advertise a service name

Format

- ANSI C, C++

```
#include <xatmi.h>
int tpadvertise (char *svcname,
                 void (*func)(TPSVCINFO *) )
```

- K&R C

```
#include <xatmi.h>
int tpadvertise (svcname, func)
char  *svcname;
void  (*func)();
```

Description

The function `tpadvertise()` allows a server to advertise the services that it offers. By default, a server's services are advertised when it is booted and unadvertised when it is shut down.

The function `tpadvertise()` advertises `svcname` for the server. The argument `svcname` should be 15 characters or fewer, but cannot be NULL or the NULL string (""). Longer names are accepted and truncated to 15 characters. Users should make sure that truncated names do not match other service names. The argument `func` is the address of a service function. This function is invoked whenever a request for `svcname` is received by the server. The argument `func` cannot be NULL.

If `svcname` is already advertised for the server and `func` matches its current function, `tpadvertise()` returns success (this includes truncated names that match already advertised names). However, if `svcname` is already advertised for the server but `func` does not match its current function, an error is returned (this can happen if truncated names match already advertised names).

<<Arguments>>

- <<`svcname`

  Specify the name of the service to be requested. >>

- <<`(*func)()`

  The address of the service function.>>

Return value

The function `tpadvertise()` returns `-1` on error and sets `tperrno` to indicate the

error condition.

■ Errors

Under the following conditions, `tpadvertise()` fails and sets `tperrno` to one of the following values:

Return value	Return value (numeric)	Explanation
TPEINVAL	4	The argument `svcname` is NULL or the NULL string (""), or `func` is NULL.
TPELIMIT	5	The argument `svcname` cannot be advertised because of space limitations.
TPEMATCH	23	The argument `svcname` is already advertised for the server, but not with a function indicated by `func`. Although the function fails, `svcname` remains advertised with its current function (that is, `func` does not replace the current function).
TPEPROTO	9	`tpadvertise()` was called in an improper context.
TPESYSTEM	12	A communication resource manager system error has occurred. The exact nature of the error is determined in a product-specific manner.
TPEOS	7	An operating system error has occurred. The exact nature of the error is determined in a product-specific manner.

See also

`tpservice()`, `tpunadvertise()`.

<<Notes on use with OpenTP1>>

1. <<The function `tpadvertise()` can be called only by SPPs. When the server starts, all services specified in the user service definition are automatically advertised. Combinations of service names and functions can be advertised only when they are specified in the user service definition of this function.>>
2. <<Under the OpenTP1, if the service group of UAPs which call the function `tpadvertise()` is the same as the service group of UAPs which have advertised the services, this function returns normally. If the two groups do not match, the function returns with an error.>>
3. <<The behavior caused by XATMI errors encountered during OSI TP communication may be different from the behavior caused by errors encountered conventional TCP/IP communication.>>

tpalloc - Allocate a typed buffer

Format

- ANSI C, C++

```
#include <xatmi.h>
char *tpalloc (char *type, char *subtype, long size)
```

- K&R C

```
#include <xatmi.h>
char  *tpalloc (type, subtype, size)
char  *type;
char  *subtype;
long  size
```

Description

The function `tpalloc()` returns a pointer to a buffer of type `type`. Depending on the type of buffer, both `subtype` and `size` are optional.

If multiple `subtypes` are available for a particular buffer type, `subtype` must be specified when `tpalloc()` is called. If the `type` specified does not have a `subtype`, `*subtype` is ignored (and may be null). The allocated buffer is at least as large as `size`.

Note that only the first eight bytes of `type` and the first 16 bytes of `subtype` are significant.

Because some buffer types require initialization before they can be used, `tpalloc()` initializes a buffer (in a communication-resource-manager-specific manner) after it is allocated and before it is returned. Thus, the buffer returned to the caller is ready for use. Note that unless the initialization processing cleared the buffer, the buffer is not initialized to zeros by `tpalloc()`.

<<Arguments>>

- `<<type`

  Specify the type name.>>

- `<<subtype`

  Specify the subtype name.>>

- `<<size`

  Specify the size of the buffer to be allocated.>>

Return value

Upon successful completion, `tpalloc()` returns a pointer to a buffer of the appropriate type aligned on a long word. Otherwise it returns NULL and sets `tperrno` to indicate the error condition.

- Errors

  Under the following conditions, `tpalloc()` fails and sets `tperrno` to one of the following values:

Return value	Return value (numeric)	Explanation
`TPEINVAL`	4	Invalid arguments were given (for example, `type` is NULL).
`TPENOENT`	6	The value for `type` or `subtype` does not exist.
`TPEPROTO`	9	`tpalloc()` was called in an improper context.
`TPESYSTEM`	12	A communication resource manager system error has occurred. The exact nature of the error is determined in a product-specific manner.
`TPEOS`	7	An operating system error has occurred. The exact nature of the error is determined in a product-specific manner.

Application usage

If buffer initialization processing fails, the allocated buffer is freed and `tpalloc()` fails returning NULL.

This function should not be used in concert with `malloc()`, `realloc()` or `free()` in the C library (for example, a buffer allocated with `tpalloc()` should not be freed with `free()`).

See also

`tpfree()`, `tprealloc()`, `tptypes()`.

<<Notes on use with OpenTP1>>

1. <<Under the OpenTP1, the buffer returned by the function `tpalloc()` is initialized to 0.>>
2. <<The behavior caused by XATMI errors encountered during OSI TP communication may be different from the behavior caused by errors encountered conventional TCP/IP communication.>>

tpcall - Send a service request and synchronously await its reply

Format

- ANSI C, C++

```
#include <xatmi.h>
int tpcall (char *svc, char *idata, long ilen,
             char **odata, long *olen, long flags)
```

- K&R C

```
#include <xatmi.h>
int tpcall (svc, idata, ilen, odata, olen, flags)
char      *svc;
char      *idata;
long      ilen;
char      **odata;
long      *olen;
long      flags;
```

Description

The function `tpcall()` sends a request and synchronously awaits its reply. A call to this function is the same as calling `tpacall()` immediately followed by `tpgetrply()`. The function `tpcall()` sends a request to the service named by `svc`. The data portion of a request is pointed to by `idata`, a buffer previously allocated by `tpalloc()`. The argument `ilen` specifies how much of `idata` to send. Note that if `idata` points to a buffer of a type that does not require a length to be specified, `ilen` is ignored (and may be 0). If `data` points to a buffer that does require a length, `len` must not be zero. Also, `idata` may be NULL in which case `ilen` is ignored. The type and sub-type of `idata` must match one of the types and sub-types recognized by `svc`.

`odata` is the address of a pointer to the buffer where a reply is read into, and the length of that reply is returned in `*olen`. `*odata` must point to a buffer originally allocated by `tpalloc()`. If the same buffer is to be used for both sending and receiving, `odata` should be set to the address of `idata`. To determine whether a reply buffer changed in size, compare its (total) size before `tpcall()` was issued with `*olen`. If `*olen` is larger, then the buffer has grown; otherwise, the buffer has not changed size. Also, if `idata` and `*odata` were equal when `tpcall()` was invoked, and `*odata` is changed, `idata` no longer points to a valid address. Note that `*odata` may change for reasons other than the buffer's size increased. If `*olen` is 0 upon return, then the reply has no data portion and neither `*odata` nor the buffer it points to were modified. It is an error for `*odata` or `olen` to be NULL.

<<Arguments>>

- <<svc

  Specify the name of the service to be requested.>>

- <<idata

  Specify the pointer to the send buffer.>>

- <<ilen

  Specify the length of the send buffer.>>

- <<odata

  Specify the address of the pointer to the buffer which will contain reply data.>>

- <<olen

  Indicates the pointer to the long-type data giving the length of the reply buffer.>>

- <<flags>>

  The valid flags are as follows:

  TPNOTRAN

  If the caller is in transaction mode and this flag is set, when svc is invoked, it is not performed on behalf of the caller's transaction. If svc does not support transactions, this flag must be set when the caller is in transaction mode. A caller in transaction mode that sets this flag is still subject to the transaction timeout (and no other). If a service fails that was invoked with this flag, the caller's transaction is not affected.

  TPNOCHANGE

  By default, if a buffer is received that differs in type from the buffer pointed to by *odata, *odata's buffer type changes to the received buffer's type so long as the receiver recognizes the incoming buffer type. When this flag is set, the type of the buffer pointed to by *odata is not allowed to change. That is, the type and sub-type of the received buffer must match the type and sub-type of the buffer pointed to by *odata.

  TPNOBLOCK

  The request is not sent if a blocking condition exists (for example, the internal buffers into which the message is transferred are full). Note that this flag applies only to the send portion of tpcall(); the function may block waiting for the reply. When TPNOBLOCK is not specified and a blocking condition exists, the caller blocks until the condition subsides or a timeout occurs (either transaction or blocking timeout).

  TPNOTIME

This flag signifies that the caller is willing to block indefinitely and wants to be immune to blocking timeouts. Transaction timeouts may still occur.

`TPSIGRSTRT`

If a signal interrupts any underlying system calls, the interrupted system call is reissued.

Return value

Upon successful return from `tpcall()` or upon return where `tperrno` is set to `TPESVCFAIL`, the `tpurcode` global contains an application-defined value that was sent as part of `tpreturn()`. Otherwise, it returns -1 and sets `tperrno` to indicate the error condition.

- Errors

Under the following conditions, `tpcall()` fails and sets `tperrno` to one of the values below. Unless otherwise noted, failure does not affect the caller's transaction, if one exists.

Return value	Return value (numeric)	Explanation
`TPEINVAL`	4	Invalid arguments were given (for example, `svc` is NULL or the value of `flags` is invalid).
`TPENOENT`	6	Cannot establish a connection because the service specified in `svc` does not exist.
`TPEITYPE`	17	`type` and `subtype` for `idata` are not in a format that can be used for `svc`.
`TPEOTYPE`	18	Either `type` and `subtype` of the reply are not known to the caller, or `TPNOCHANGE` was set in flags, but the buffer `type` and `subtype` specified for `*odata` do not match `type` and `subtype` of the reply sent by the service. If this error occurs, neither `*odata` nor `*olen` is changed. If the service request was made as the caller's current transaction, the transaction is marked `rollback_only` since the reply is discarded.
`TPETRAN`	14	`TPNOTRAN` was not set, even though transaction processing could not be performed for `svc`.
`TPETIME`	13	A timeout occurred. If the caller is in transaction mode, a transaction time-out occurred and the transaction is marked `rollback_only`; otherwise, a blocking time-out occurred and neither `TPNOBLOCK` nor `TPNOTIME` were specified. In either case, neither `*odata` nor `*olen` is changed. If a transaction time-out occurred, any attempts to send new requests or receive outstanding replies fail with `TPETIME` until the transaction has been rolled back.

Return value	Return value (numeric)	Explanation
TPESVCFAIL	11	The service function sending the caller's reply called tpreturn() with TPFAIL. This is an application-level failure. The contents of the reply sent by the service are available in the buffer pointed to by *odata. If the service request was made as the caller's current transaction, the transaction is marked rollback_only. *Note* So long as the transaction has not timed out, further communication may be performed up until before rollback. In this case, any work performed as the caller's transaction is rolled back upon transaction completion. Be sure to set TPNOTRAN for communication with continuous processing enabled. Depending on the transaction function, some processing is performed to rollback the caller's transaction.
TPESVCERR	10	This error was encountered either in invoking a service function or during its completion in tpreturn() (for example, bad arguments were passed). No reply data is returned when this error occurs (that is, neither *odata, nor *olen are changed). If the reply processing for the service request was made as the caller's transaction, the transaction is marked the rollback_only status. *Note* So long as the transaction has not timed out, further communication may be attempted before rolling back the transaction. Such attempts may be processed normally or may fail (producing an error return or event). Be sure to set TPNOTRAN for communication with continuous processing enabled. Depending on the transaction function, some processing is performed to rollback the caller's transaction.
TPEBLOCK	3	When tpcall() for which TPNOBLOCK was specified was called, the blocking status existed.
TPEGOTSIG	15	A signal was received, but TPSIGRSTRT was not set.
TPEPROTO	9	tpcall() was called in an improper context.
TPESYSTEM	12	A communication resource manager system error has occurred. The exact nature of the error is determined in a product-specific manner.
TPEOS	7	An operating system error has occurred. The exact nature of the error is determined in a product-specific manner.

See also

`tpalloc()`, `tpacall()`, `tpgetrply()`, `tpreturn()`.

<<Notes on use with OpenTP1>>

1. <<The `TPNOBLOCK` flag is invalid under the relevant version of the OpenTP1. Therefore, the error code `TPEBLOCK` will not be returned to `tperrno`. The OpenTP1 is designed so that if communication is impossible because of blocking, `TPESYSTEM` is returned as when communication is impossible because of network failure.>>
2. <<Under the relevant version of the OpenTP1, the `TPNOTIME` flag is valid only when a reply is received. It is invalid when blocking occurs at the time of request sending.>>
3. <<The `TPSIGRSTRT` flag is invalid. Regardless of this flag, when a signal is received, the interrupted system call is reinvoked. `TPEGOTSIG` will never return.>>
4. <<Under the relevant version of the OpenTP1, `TPEITYPE` will not return. If `data` of a type unavailable with `svc` is passed, `TPESYSTEM` will return. If the calling program is in transaction mode, the `rollback_only` state comes into effect.>>
5. <<Under the OpenTP1, when a process encounters transaction timeout, it terminates abnormally. Therefore, `TPETIME` returns only when blocking timeout occurs.>>
6. <<Under the relevant version of the OpenTP1, data which requires rollback causes the return of `TPESYSTEM` unless otherwise specified by the X/Open. However, the `rollback_only` state may not come into effect even when `TPESYSTEM` returns.>>
7. <<If an SPP to which the service request is addressed terminates abnormally, the function may return with a `TPETIME` error before the time specified for `watch_time` in the definition has elapsed. If 0 (wait until response reception) is specified for `watch_time`, the function may return with a `TPEPROTO` error.>>
8. <<The function returns with a `TPEPROTO` error if the OpenTP1 security facility is used but the service request is not authenticated. Whether the cause of the error return is because the service request is not authenticated can be checked with the UAP trace detail error code.>>
9. <<For OSI TP communication using TP1/NET/OSI-TP-Extended, a line failure forces control to return, and outputs `TPESVCERR`.>>
10. <<For OSI TP communication using TP1/NET/OSI-TP-Extended, transmission data must not exceed the length specified in the length operand of the NET buffer group definition `nettbuf` (NET/Library common definition).>>

11. <<During OSI TP communication, the following conditions cause a TPESVCERR error when an attempt is made to issue the function tpcall() or tpgetrply(); during TCP/IP communication, they cause a TPENOENT or TPESYSTEM error when the same attempt is made:
    - The specified service does not exist at the request destination.
    - The typed buffer is not recognized by the server.
    - Service activation encounters an error.>>
12. <<If the number of system associations is insufficient during OSI TP communication, the function outputs a log message and returns with TPESYSTEM.>>
13. <<While OSI TP communication is in use, blocking time-out occurs even if TPNOTIME is specified. While TCP/IP communication is in use, blocking time-out occurs during non-transaction periods.>>
14. <<For OSI TP communication, the value assigned to the user service definition message_store_buflen must be equal to or greater than the size specified by nettbuf -g. For TCP/IP communication, the same rules as for the function dc_rpc_call() apply.>>
15. <<Suppose that inter-TP1 OSI TP communication is in use. When a service with N specified for the atomic_update clause is called as a transaction, TPESVCERR is returned to the service requester.>>
16. <<Suppose that a service called via TCP/IP communication calls a service via OSI TP communication and that the service function ends without receiving a response. The UAP which called the service via TCP/IP communication receives a normal response message. The functions tpcall() and tpgetrply() return normally.>>
17. <<The behavior caused by XATMI errors encountered during OSI TP communication may be different from the behavior caused by errors encountered conventional TCP/IP communication.>>

tpcancel - Cancel a call descriptor for an outstanding reply

Format

- ANSI C, C++

```
#include <xatmi.h>
int tpcancel (int cd)
```

- K&R C

```
#include <xatmi.h>
int tpcancel (cd)
int     cd;
```

Description

The function `tpcancel()` cancels a call descriptor, `cd`, returned by `tpacall()`. It is an error to attempt to cancel a call descriptor associated with a global transaction.

Upon successful return, `cd` is no longer valid and any reply received (by the communication resource manager) on behalf of `cd` is silently discarded.

<<Argument>>

- <<`cd`

  Specify a descriptor.>>

Return value

`tpcancel()` returns `-1` on error and sets `tperrno` to indicate the error condition.

- Errors

  Under the following conditions, `tpcancel()` fails and sets `tperrno` to one of the following values:

Return value	Return value (numeric)	Explanation
`TPEBADDESC`	2	The argument `cd` is an invalid descriptor.
`TPETRAN`	14	The argument `cd` is associated with the caller's global transaction. Even after an error, the descriptor `cd` remains valid and the caller's current transaction is not affected.
`TPEPROTO`	9	The function `tpcancel()` was called in an improper context.

Return value	Return value (numeric)	Explanation
TPESYSTEM	12	A communication resource manager system error has occurred. The exact nature of the error is determined in a product-specific manner.
TPEOS	7	An operating system error has occurred. The exact nature of the error is determined in a product-specific manner.

See also

tpacall().

<<Notes on use with OpenTP1>>

1. <<The behavior caused by XATMI errors encountered during OSI TP communication may be different from the behavior caused by errors encountered conventional TCP/IP communication.>>

tpconnect - Establish a conversational service connection

Format

- ANSI C, C++

```
#include <xatmi.h>
int tpconnect (char *svc, char *data, long len,
               long flags)
```

- K&R C

```
#include <xatmi.h>
int tpconnect (svc, data, len, flags)
char      *svc;
char      *data;
long      len;
long      flags;
```

Description

The function `tpconnect()` allows a program to set up a half-duplex connection to a conversational service, `svc`.

As part of setting up a connection, the caller can pass application-defined data to the receiving service routine. If the caller chooses to pass data, `data` must point to a buffer previously allocated by `tpalloc()`. `len` specifies how much of the buffer to send. Note that if `data` points to a buffer of a type that does not require a length to be specified, `len` is ignored (and may be 0). If `data` points to a buffer that does require a length, `len` must not be zero. Also, `data` can be NULL in which case `len` is ignored (no application data is passed to the conversational service). The type and sub-type of `data` must match one of the types and sub-types recognized by `svc`. Because the conversational service receives `data` and `len` via the `TPSVCINFO` structure upon invocation, the service does not call `tprecv()` to get the data sent by `tpconnect()`.

<<Arguments>>

- <<`svc`

  Specify the name of the service to be requested.>>

- <<`data`

  Specify the pointer to the send data storage area.>>

- <<`len`

  Specify the length of the send data.>>

- <<flags>>

  The valid flags are as follows:

  TPNOTRAN

  If the caller is in transaction mode and this flag is set, when svc is invoked, it is not performed on behalf of the caller's transaction. If svc does not support transactions, this flag must be set when the caller is in transaction mode. A caller in transaction mode that sets this flag is still subject to the transaction timeout (and no other). If a service fails that was invoked with this flag, the caller's transaction is not affected.

  TPSENDONLY

  The caller wants the connection to be set up initially such that it can send data and the called service can only receive data (that is, the caller initially has control of the connection). Either TPSENDONLY or TPRECVONLY must be specified.

  TPRECVONLY

  The caller wants the connection to be set up initially such that it can only receive data and the called service can send data (that is, the service being called initially has control of the connection). Either TPSENDONLY or TPRECVONLY must be specified.

  TPNOBLOCK

  The connection is not established and the data is not sent if a blocking condition exists (for example, the internal buffers into which the message is transferred are full). When TPNOBLOCK is not specified and a blocking condition exists, the caller blocks until the condition subsides or a timeout occurs (either transaction or blocking timeout).

  TPNOTIME

  This flag signifies that the caller is willing to block indefinitely and wants to be immune to blocking timeouts. Transaction timeouts may still occur.

  TPSIGRSTRT

  If a signal interrupts any underlying system calls, the interrupted system call is reissued.

Return value

Upon successful completion, tpconnect() returns a descriptor that is used to refer to the connection in subsequent calls. Otherwise it returns -1 and sets tperrno to indicate the error condition.

- Errors

Under the following conditions, tpconnect() fails and sets tperrno to one of the

values below. Unless otherwise noted, failure does not affect the caller's transaction, if one exists.

Return value	Return value (numeric)	Explanation
TPEINVAL	4	Invalid arguments were given (for example, svc is NULL, data is non-NULL and does not point to a buffer allocated by tpalloc(), TPSENDONLY or TPRECVONLY was not specified in flags, or the value of flags is invalid).
TPENOENT	6	Cannot establish a connection because the service specified in svc does not exist.
TPEITYPE	17	The type and subtype for data are not in a format that can be used for svc.
TPELIMIT	5	The caller's request was not sent because the maximum number of outstanding connections has been reached.
TPETRAN	14	TPNOTRAN was not set, even though transaction processing could not be performed for svc.
TPETIME	13	A timeout occurred. If the caller is in transaction mode, a transaction time-out occurred and the transaction is marked rollback_only; otherwise, a blocking time-out occurred and neither TPNOBLOCK nor TPNOTIME were specified. If a transaction time-out occurred, any attempts to send or receive messages on any connections or to start a new connection fail with TPETIME until the transaction has been rolled back.
TPEBLOCK	3	When tpconnect() for which TPNOBLOCK was specified was called, the blocking status existed.
TPEGOTSIG	15	A signal was received, but TPSIGRSTRT was not set.
TPEPROTO	9	tpconnect() was called in an improper context.
TPESYSTEM	12	A communication resource manager system error has occurred. The exact nature of the error is determined in a product-specific manner.
TPEOS	7	An operating system error has occurred. The exact nature of the error is determined in a product-specific manner.

See also

tpalloc(), tpdiscon(), tprecv(), tpsend(), tpservice().

<<Notes on use with OpenTP1>>

1. <<The TPNOBLOCK flag is invalid under the relevant version of the OpenTP1.

Therefore, the error code TPEBLOCK will not be returned to tperrno. The OpenTP1 is designed so that if communication is impossible because of blocking, TPESYSTEM is returned as when communication is impossible because of network failure.>>

2. <<The TPNOTIME flag is invalid under the relevant version of the OpenTP1.>>
3. <<The TPSIGRSTRT flag is invalid. Regardless of this flag, when a signal is received, the interrupted system call is reinvoked. TPEGOTSIG will never return.>>
4. <<Under the relevant version of the OpenTP1, TPEITYPE will not return. If data of a type unavailable with svc is passed, TPESYSTEM will return. If the calling program is in transaction mode, the rollback_only state comes into effect.>>
5. <<Under the OpenTP1, when a process encounters transaction timeout, it terminates abnormally. Therefore, TPETIME returns only when blocking timeout occurs.>>
6. <<Under the relevant version of the OpenTP1, data which requires rollback causes the return of TPESYSTEM unless otherwise specified by the X/Open. However, the rollback_only state may not come into effect even when TPESYSTEM returns.>>
7. <<The function returns with a TPEPROTO error if the OpenTP1 security facility is used but the service request is not authenticated. Whether the cause of the error return is because the service request is not authenticated can be checked with the UAP trace detail error code.>>
8. <<For OSI TP communication using TP1/NET/OSI-TP-Extended, conversational service communication cannot be held. If this is done, the system operation is undefined.>>
9. <<If the server AP is in shutdown status, the system operates as follows depending on whether the request destination SPP that is shutdown is on a local node or on a remote node:

   When the request destination SPP on a local node is shutdown:

   tpconnect() returns -1 and sets the value TPEPROTO in tperrno.

   When the request destination SPP on a remote node is shutdown:

   In the transaction mode, the server AP terminates abnormally due to transaction time-out.

   In the non-transaction mode, tpconnect() returns -1 and sets the value TPETIME in tperrno.>>

tpdiscon - Terminate a conversational service connection abortively

Format

- ANSI C, C++

```
#include <xatmi.h>
int tpdiscon (int cd)
```

- K&R C

```
#include <xatmi.h>
int tpdiscon (cd)
int    cd;
```

Description

The function `tpdiscon()` immediately terminates the connection specified by `cd` and generates a `TPEV_DISCONIMM` event on the other end of the connection.

The function `tpdiscon()` can be called only by the originator of the conversation. `tpdiscon()` cannot be called within a conversational service on the descriptor with which it was invoked. Rather, a conversational service must use `tpreturn()` to signify that it has completed its part of the conversation. Similarly, even though a program communicating with a conversational service can issue `tpdiscon()`, the preferred way is to let the service terminate the connection in `tpreturn()`; doing so ensures correct results.

The function `tpdiscon()` causes the connection to be terminated immediately (that is, abortively rather than orderly). Any data that has not yet reached its destination may be lost. `tpdiscon()` can be issued even when the program on the other end of the connection is participating in the caller's transaction. In this case, the transaction must be rolled back. Also, the caller does not need to have control of the connection when `tpdiscon()` is called.

<<Argument>>

- <<`cd`

  Specify a descriptor.>>

Return value

The function `tpdiscon()` returns `-1` on error and sets `tperrno` to indicate the error condition.

- Errors

  Under the following conditions, `tpdiscon()` fails and sets `tperrno` to one of the

following values:

Return value	Return value (numeric)	Explanation
TPEBADDESC	2	The argument cd is invalid, or is the descriptor with which a conversational service was invoked.
TPETIME	13	A timeout occurred. The descriptor is no longer valid.
TPEPROTO	9	The function tpdiscon() was called in an improper context.
TPESYSTEM	12	A communication resource manager system error has occurred. The exact nature of the error is determined in a product-specific manner.
TPEOS	7	An operating system error has occurred. The exact nature of the error is determined in a product-specific manner.

See also

tpconnect(), tprecv(), tpreturn(), tpsend().

<<Notes on use with OpenTP1>>

1. <<The error code TPETIME will not be returned to tperrno under the relevant version of the OpenTP1.>>
2. <<For OSI TP communication using TP1/NET/OSI-TP-Extended, conversational service communication cannot be held. If this is done, the system operation is undefined.>>

tpfree - Free a typed buffer

Format

- ANSI C, C++

```
#include <xatmi.h>
void tpfree (char *ptr)
```

- K&R C

```
#include <xatmi.h>
void tpfree (ptr)
char    *ptr;
```

Description

The argument to `tpfree()` is a pointer to a buffer previously obtained by either `tpalloc()` or `tprealloc()`. If `ptr` is NULL, no action occurs. Undefined results occur if `ptr` does not point to a typed buffer (or if it points to space previously freed with `tpfree()`). Inside service routines, `tpfree()` returns and does not free the buffer if `ptr` points to the buffer passed into a service routine.

Some buffer types require state information or associated data to be removed as part of freeing a buffer. `tpfree()` removes any of these associations (in a communication-resource-manager-specific manner) before a buffer is freed.

Once `tpfree()` returns, `ptr` should not be passed as an argument to any XATMI routine or used in any other manner.

<<Argument>>

- <<`ptr`

  Specify the pointer to the buffer allocated by the function `tpalloc()` or `tprealloc()`.>>

Return value

The function `tpfree()` does not return any value to its caller. Therefore, it is declared as a void.

Application usage

This function should not be used in concert with `malloc()`, `realloc()` or `free()` in the C library (for example, a buffer allocated with `tpalloc()` should not be freed with `free()`).

See also

`tpalloc()`, `tprealloc()`.

<<Notes on use with OpenTP1>>

1. <<The behavior caused by XATMI errors encountered during OSI TP communication may be different from the behavior caused by errors encountered conventional TCP/IP communication.>>

tpgetrply - Get a reply from a previous service request

Format

- ANSI C, C++

```
#include <xatmi.h>
int tpgetrply (int *cd, char **data, long *len, long
               flags)
```

- K&R C

```
#include <xatmi.h>
int tpgetrply (cd, data, len, flags)
int      *cd;
char     **data;
long     *len;
long     flags;
```

Description

The function `tpgetrply()` returns a reply from a previously-sent service request. This function's first argument, `cd`, points to a call descriptor returned by `tpacall()`. By default, the function waits until the reply matching `*cd` arrives or a timeout occurs.

`data` must be the address of a pointer to a buffer previously allocated by `tpalloc()` and `len` should point to a `long` that `tpgetrply()` sets to the amount of data successfully received. `tpgetrply()` ensures that the request fits into the specified buffer by growing the buffer if necessary. Upon successful return, `*data` points to a buffer containing the reply and `*len` contains the size of the data. Note that `*data` may have changed upon return for reasons other than an increase in the size of the buffer. If `*len` is greater than the total size of the buffer before the call, the buffer's new size is `*len`. If `*len` is `0`, then the reply dequeued has no data portion and neither `*data` nor the buffer it points to were modified. It is an error for `*data` or `len` to be NULL.

<<Arguments>>

- <<`cd`

  Specify a descriptor.>>

- <<`data`

  Specify the address of the pointer to the buffer which will contain received data.>>

- <<`len`

  Specify the address of the area which will contain the length of received data.>>

- <<flags>>

  The valid flags are as follows:

  TPGETANY

  This flag signifies that tpgetrply() should ignore the descriptor pointed to by cd, return any reply available and set cd to point to the call descriptor for the reply returned. If no replies exist, by default tpgetrply() waits for one to arrive.

  TPNOCHANGE

  By default, if a buffer is received that differs in type from the buffer pointed to by *data, then *data's buffer type changes to the received buffer's type so long as the receiver recognizes the incoming buffer type. When this flag is set, the type of the buffer pointed to by *data is not allowed to change. That is, the type and sub-type of the received buffer must match the type and sub-type of the buffer pointed to by *data.

  TPNOBLOCK

  tpgetrply() does not wait for the reply to arrive. If the reply is available, tpgetrply() gets the reply and returns. When this flag is not specified and a reply is not available, the caller blocks until the reply arrives or a timeout occurs (either transaction or blocking timeout).

  TPNOTIME

  This flag signifies that the caller is willing to block indefinitely for its reply and wants to be immune to blocking timeouts. Transaction timeouts may still occur.

  TPSIGRSTRT

  If a signal interrupts any underlying system calls, the interrupted system call is reissued.

  Except as noted below, *cd is no longer valid after its reply is received.

Return value

Upon successful return from tpgetrply() or upon return where tperrno is set to TPESVCFAIL, the tpurcode global contains an application-defined value that was sent as part of tpreturn(). Otherwise, it returns -1 and sets tperrno to indicate the error condition.

- Errors

  Under the following conditions, tpgetrply() fails and sets tperrno as indicated below. Note that if TPGETANY is not set, *cd is invalidated unless otherwise stated. If TPGETANY is set, cd points to the descriptor for the reply on which the failure occurred; if an error occurred before a reply could be retrieved, cd points to 0. Also, the failure does not affect the caller's transaction, if one exists, unless otherwise stated.

Return value	Return value (numeric)	Explanation
TPEINVAL	4	Invalid arguments were given (for example, cd, data, *data or len is NULL or the value of flags is invalid). If cd is non-NULL, it is still valid after this error and the reply remains unresolved.
TPEBADDESC	2	The argument cd points to an invalid descriptor.
TPEOTYPE	18	Either type and subtype of the reply are not known to the caller, or TPNOCHANGE was set in flags, but the buffer type and subtype specified for *data do not match type and subtype of the reply sent by the service. If this error occurs, neither *data nor *len is changed. If the reply was to be received as the caller's current transaction, the transaction is marked rollback_only since the reply is discarded.
TPETIME	13	A timeout occurred. If the caller is in transaction mode, a transaction time-out occurred and the transaction is marked rollback_only; otherwise, a blocking time-out occurred and neither TPNOBLOCK nor TPNOTIME were specified. In either case, neither *data nor *len is changed (unless the caller is in transaction mode). The argument *cd remains valid (and TPGETANY was not set). If a transaction time-out occurred, any attempts to send new requests or receive outstanding replies fail with TPETIME until the transaction has been rolled back.
TPESVCFAIL	11	The service function sending the caller's reply called tpreturn() with TPFAIL. This is an application-level failure. The contents of the reply sent by the service are available in the buffer pointed to by *data. If the service request was made as the caller's current transaction, the transaction is marked rollback_only. *Note* So long as the transaction has not timed out, further communication may be performed up until before rollback. In this case, any work performed as the caller's transaction is rolled back upon transaction completion. Be sure to set TPNOTRAN for communication with continuous processing enabled. Depending on the transaction function, some processing is performed to rollback the caller's transaction.

Return value	Return value (numeric)	Explanation
TPESVCERR	10	An error was encountered either in invoking a service function or during its completion in tpreturn() (for example, bad arguments were passed). No reply data is returned when this error occurs (that is, neither *data, nor *len is changed). If the reply was received as the caller's transaction, the transaction is marked rollback_only. *Note* So long as the transaction has not timed out, further communication may be attempted before completely rolling back the transaction. Such attempts may be processed normally or may fail (producing an error return or event). Be sure to set TPNOTRAN for communication with continuous processing enabled. Depending on the transaction function, some processing is performed to rollback the caller's transaction.
TPEBLOCK	3	When TPNOBLOCK was specified, the blocking status existed. The argument *cd remains valid.
TPEGOTSIG	15	A signal was received, but TPSIGRSTRT was not set.
TPEPROTO	9	tpgetrply() was called in an improper context.
TPESYSTEM	12	A communication resource manager system error has occurred. The exact nature of the error is determined in a product-specific manner.
TPEOS	7	An operating system error has occurred. The exact nature of the error is determined in a product-specific manner.

See also

tpacall(), tpalloc(), tpreturn().

<<Notes on use with OpenTP1>>

1. <<The TPSIGRSTRT flag is invalid. Regardless of this flag, when a signal is received, the interrupted system call is reinvoked. TPEGOTSIG will never return.>>
2. <<Under the OpenTP1, when a process encounters transaction timeout, it terminates abnormally. Therefore, TPETIME returns only when blocking timeout occurs.>>
3. <<Under the relevant version of the OpenTP1, data which requires rollback causes the return of TPESYSTEM unless otherwise specified by the X/Open. However, the rollback_only state may not come into effect even when TPESYSTEM returns.>>

4. <<If the function `tpacall()` passes data of a type that cannot be used by the called service, it returns normally, but the function `tpgetrply()` will encounter an error. If the function `tpgetrply()` encounters a `TPESYSTEM` error, check the results of the function `tpacall()` as well.>>
5. <<If an SPP to which a service was requested terminates abnormally, the function might return with a `TPETIME` error before the time specified in the `watch_time` operand in the definition has elapsed. If `0` (wait until a response is received) is specified in the `watch_time` operand, the function might return with a `TPEPROTO` error.>>
6. <<The function returns with a `TPEPROTO` error if the OpenTP1 security facility is used but the service request is not authenticated. Whether the cause of the error return is because the service request is not authenticated can be checked with the UAP trace detail error code.>>
7. <<For OSI TP communication using TP1/NET/OSI-TP-Extended, receive data must not exceed the length specified in the length operand of the NET buffer group definition `nettbuf` (NET/Library common definition).>>
8. <<Suppose that inter-TP1 OSI TP communication is in use. When a service with N specified for the `atomic_update` clause is called as a transaction, `TPESVCERR` is returned to the service requester.>>
9. <<Suppose that a service called via TCP/IP communication calls a service via OSI TP communication and that the service function ends without receiving a response. The UAP which called the service via TCP/IP communication receives a normal response message. The functions `tpcall()` and `tpgetrply()` return normally.>>
10. <<The behavior caused by XATMI errors encountered during OSI TP communication may be different from the behavior caused by errors encountered conventional TCP/IP communication.>>

tprealloc - Change the size of a typed buffer

Format

- ANSI C, C++

```
#include <xatmi.h>
char *tprealloc (char *ptr, long size)
```

- K&R C

```
#include <xatmi.h>
char *tprealloc (ptr, size)
char      *ptr;
long      size;
```

Description

The function `tprealloc()` changes the size of the buffer pointed to by `ptr` to `size` bytes and returns a pointer to the new (possibly moved) buffer. As with `tpalloc()`, the size of the buffer is at least as large as `size`. A buffer's type remains the same after it is reallocated. After this function returns successfully, the returned pointer should be used to reference the buffer; `ptr` should no longer be used. The buffer's contents do not change up to the lesser of the new and old sizes.

Some buffer types require initialization before they can be used. `tprealloc()` reinitializes a buffer (in a communication-resource-manager-specific manner) after it is reallocated and before it is returned. Thus, the buffer returned to the caller is ready for use.

<<Arguments>>

- <<`ptr`

  Specify the pointer to the buffer.>>

- <<`size`

  Specify the size which will be in effect after the buffer is reallocated.>>

Return value

Upon successful completion, `tprealloc()` returns a pointer to a buffer of the appropriate type aligned on a long word. Otherwise it returns NULL and sets `tperrno` to indicate the error condition.

- Errors

  Under the following conditions, `tprealloc()` fails and sets `tperrno` to one of the following values:

Return value	Return value (numeric)	Explanation
TPEINVAL	4	Invalid arguments were given (for example, ptr is not a point to a buffer allocated for tpalloc()).
TPEPROTO	9	tprealloc() was called in an improper context.
TPESYSTEM	12	A communication resource manager system error has occurred. The exact nature of the error is determined in a product-specific manner.
TPEOS	7	An operating system error has occurred. The exact nature of the error is determined in a product-specific manner.

Application usage

If buffer reinitialization fails, tprealloc() fails returning NULL and the contents of the buffer pointed to by ptr may not be valid.

This function should not be used in concert with malloc(), realloc() or free() in the C library (for example, a buffer allocated with tprealloc() should not be freed with free()).

See also

tpalloc(), tpfree(), tptypes().

<<Notes on use with OpenTP1>>

1. <<Under the OpenTP1, the buffer returned by the function tprealloc() is reinitialized to 0.>>
2. <<The behavior caused by XATMI errors encountered during OSI TP communication may be different from the behavior caused by errors encountered conventional TCP/IP communication.>>

tprecv - Receive a message in a conversational connection

Format

- ANSI C, C++

```
#include <xatmi.h>
int tprecv (int cd, char **data, long *len, long flags,
            long *revent)
```

- K&R C

```
#include <xatmi.h>
int tprecv (cd, data, len, flags, revent)
int      cd;
char     **data;
long     *len;
long     flags;
long     *revent;
```

Description

The function `tprecv()` is used to receive data sent across an open connection from another program. This function's first argument, `cd`, specifies on which open connection to receive data. `cd` is a descriptor returned from either `tpconnect()` or the `TPSVCINFO` parameter to the service. The second argument, `data`, is the address of a pointer to a buffer previously allocated by `tpalloc()`.

Upon successful return, and for several event types, `*data` points to the data received and `*len` contains the size of the buffer. Note that if `*len` is greater than the total size of the buffer before the call, then the buffer's new size is `*len`. If `*len` is 0, no data was received and neither `*data` nor the buffer it points to were modified. It is an error for data, `*data` or `len` to be NULL.

`tprecv()` can be issued only by the program that does not have control of the connection.

<<Arguments>>

- `<<cd`

  Specify a descriptor.>>

- `<<data`

  Specify the address of the pointer to the buffer which will contain received data.>>

- `<<len`

  Specify the address of the area which will contain the length of received data.>>

- <<flags

  Indicates flags.>>

- <<revent

  Indicates the pointer to the long-type data about the event.>>

  The valid flags are as follows:

  TPNOCHANGE

  By default, if a buffer is received that differs in type from the buffer pointed to by *data, then *data's buffer type changes to the received buffer's type so long as the receiver recognizes the incoming buffer type. When this flag is set, the type of the buffer pointed to by *data is not allowed to change. That is, the type and sub-type of the received buffer must match the type and sub-type of the buffer pointed to by *data.

  TPNOBLOCK

  The function tprecv() does not wait for data to arrive. If data is already available to receive, tprecv() gets the data and returns. When this flag is not specified and data is not available to receive, the caller blocks until data arrives.

  TPNOTIME

  This flag signifies that the caller is willing to block indefinitely and wants to be immune to blocking timeouts. Transaction timeouts may still occur.

  TPSIGRSTRT

  If a signal interrupts any underlying system calls, then the interrupted system call is reissued.

  If an event exists for the descriptor, cd, and tprecv() encounters no errors, the event type is returned in revent. data can be received along with the TPEV_SVCSUCC, TPEV_SVCFAIL, and TPEV_SENDONLY events. Valid events for tprecv() are as follows:

  TPEV_DISCONIMM

  Received by the subordinate of a conversation, this event indicates that the originator of the conversation has either issued an immediate disconnect on the connection by means of tpdiscon(), or it issued tpreturn(), tx_commit() or tx_rollback() with the connection still open. This event is also returned to the originator or subordinate when a connection is broken due to a communication error (for example, a server, machine, or network failure). Because this is an immediate disconnection notification (that is, abortive rather than orderly), data in transit may be lost. If the two programs were participating in the same transaction, the transaction is marked rollback_only. The descriptor used for the connection is no longer valid.

TPEV_SENDONLY

The program at the other end of the connection has relinquished control of the connection. The recipient of this event is allowed to send data but cannot receive any data until it relinquishes control.

TPEV_SVCERR

Received by the originator of a conversation, this event indicates that the subordinate of the conversation has issued tpreturn(). tpreturn() encountered an error that precluded the service from returning successfully. For example, bad arguments may have been passed to tpreturn() or it may have been called while the service had open connections to other subordinates. Due to the nature of this event, any application-defined data or return code are not available. The connection has been terminated and cd is no longer a valid descriptor. If this event occurred as part of the recipient's transaction, the transaction is marked rollback-only.

TPEV_SVCFAIL

Received by the originator of a conversation, this event indicates that the subordinate service on the other end of the conversation has finished unsuccessfully as defined by the application (that is, it called tpreturn() with TPFAIL). If the subordinate service was in control of this connection when tpreturn() was called, it can pass a typed buffer back to the originator of the connection. As part of ending the service routine, the server has terminated the connection. Thus, cd is no longer a valid descriptor. If this event occurred as part of the recipient's transaction, the transaction is marked rollback-only.

TPEV_SVCSUCC

Received by the originator of a conversation, this event indicates that the subordinate service on the other end of the conversation has finished successfully as defined by the application (that is, it called tpreturn() with TPSUCCESS). As part of ending the service routine, the server has terminated the connection. Thus, cd is no longer a valid descriptor. If the recipient is in transaction mode, it can either commit (if it is also the initiator) or roll back the transaction causing the work done by the server (if also in transaction mode) to either commit or roll back.

Return value

Upon return from tprecv() where revent is set to either TPEV_SVCSUCC or TPEV_SVCFAIL, the tpurcode global contains an application-defined value that was sent as part of tpreturn(). The function tprecv() returns -1 on error and sets tperrno to indicate the error condition. Also, if an event exists and no errors were encountered, tprecv() returns -1 and tperrno is set to TPEEVENT.

- Errors

Under the following conditions, tprecv() fails and sets tperrno to one of the

following values:

Return value	Return value (numeric)	Explanation
TPEINVAL	4	Invalid arguments were given (for example, data is not a pointer to a buffer allocated for tpalloc() or the value of flags is invalid).
TPEBADDESC	2	The argument cd points to an invalid descriptor.
TPEOTYPE	18	Either type and subtype of the incoming buffer are not known to the caller, or TPNOCHANGE was set in flags, but the type and subtype of *data do not match type and subtype of the incoming buffer. In either case, neither *data nor *len is changed. If an interactive service is executed as the caller's transaction, the transaction has the rollback_only status until the incoming buffer is discarded. When this error occurs, any event for cd is dropped and the conversation is in an undetermined status. The caller should terminate the conversation.
TPETIME	13	A timeout occurred. If the caller is in transaction mode, a transaction time-out occurred and the transaction is marked rollback_only; otherwise, a blocking time-out occurred and neither TPNOBLOCK nor TPNOTIME was specified. In either case, *data and its contents are not changed. If a transaction time-out occurred, any attempts to send or receive messages on any connections or to start a new connection fail with TPETIME until the transaction has been rolled back.
TPEEVENT	22	An event occurred and its type is returned in revent.
TPEBLOCK	3	When tprecv() for which TPNOBLOCK was specified was called, the blocking status existed.
TPEGOTSIG	15	A signal was received, but TPSIGRSTRT was not set.
TPEPROTO	9	tprecv() was called in an improper context.
TPESYSTEM	12	A communication resource manager system error has occurred. The exact nature of the error is determined in a product-specific manner.
TPEOS	7	An operating system error has occurred. The exact nature of the error is determined in a product-specific manner.

See also

tpalloc(), tpconnect(), tpdiscon(), tpsend().

<<Notes on use with OpenTP1>>

1. <<The TPSIGRSTRT flag is invalid. Regardless of this flag, when a signal is received, the interrupted system call is reinvoked. TPEGOTSIG will never return.>>
2. <<Under the OpenTP1, when a process encounters transaction timeout, it terminates abnormally. Therefore, TPETIME returns only when blocking timeout occurs.>>
3. <<Under the relevant version of the OpenTP1, data which requires rollback causes the return of TPESYSTEM unless otherwise specified by the X/Open. However, the rollback_only state may not come into effect even when TPESYSTEM returns. >>
4. <<For OSI TP communication using TP1/NET/OSI-TP-Extended, conversational service communication cannot be held. If this is done, the system operation is undefined.>>

tpreturn - Return from a service routine

Format

- ANSI C, C++

```
#include <xatmi.h>
void tpreturn (int rval, long rcode, char *data,
               long len, long flags)
```

- K&R C

```
#include <xatmi.h>
void tpreturn (rval, rcode, data, len, flags)
int       rval;
long      rcode;
char      *data;
long      len;
long      flags;
```

Description

The function `tpreturn()` indicates that a service routine has completed. `tpreturn()` acts like a return statement in the C-language (that is, when `tpreturn()` is called, the service routine returns to the communication resource manager). It is recommended that `tpreturn()` be called from within the service routine dispatched by the communication resource manager to ensure correct return of control to the communication resource manager.

The function `tpreturn()` is used to send a service's reply message. If the program receiving the reply is waiting in either `tpcall()`, `tpgetrply()`, or `tprecv()`, then after a successful call to `tpreturn()`, the reply is available in the receiver's buffer.

For conversational services, `tpreturn()` also terminates the connection. That is, the service routine cannot call `tpdiscon()` directly. To ensure correct results, the program that connected to the conversational service should not call `tpdiscon()`; rather, it should wait for notification that the conversational service has completed (that is, it should wait for one of the events, like `TPEV_SVCSUCC` or `TPEV_SVCFAIL`, sent by `tpreturn()`).

If the service routine was in transaction mode, `tpreturn()` places the service's portion of the transaction in a state where it may be either committed or rolled back when the transaction is completed. A service may be invoked multiple times as part of the same transaction so it is not necessarily fully committed nor rolled back until either `tx_commit()` or `tx_rollback()` is called by the originator of the transaction.

The function `tpreturn()` should be called after receiving all replies expected from service requests initiated by the service routine. Otherwise, depending on the nature of

the service, either a TPESVCERR error or a TPEV_SVCERR event is returned to the program that initiated communication with the service routine. Any outstanding replies that are not received are automatically dropped by the communication resource manager. In addition, the descriptors for those replies become invalid.

The function tpreturn() should be called after closing all connections initiated by the service. Otherwise, depending on the nature of the service, either a TPESVCERR or a TPEV_SVCERR event is returned to the program that initiated communication with the service routine. Also, an immediate disconnect event (that is, TPEV_DISCONIMM) is sent over all open connections to subordinates.

Concerning control of the connection, if the service routine does not have control over the connection with which it was invoked when it issues tpreturn(), two outcomes are possible. Firstly, if the service routine calls tpreturn() with rval set to TPFAIL and data is NULL, then a TPEV_SVCFAIL event is sent to the originator of this conversation. Secondly, if any other invocation of tpreturn() is used, a TPEV_SVCERR event is sent to the originator.

Since a conversational service has only one open connection that it did not initiate, the communication resource manager knows over which descriptor data (and any event) should be sent. For this reason, a descriptor is not passed to tpreturn().

The argument rval can be set to one of the following:

TPSUCCESS

The service has terminated successfully. If data is present, it is sent (barring any failures processing the return). If the caller is in transaction mode, tpreturn() places the caller's portion of the transaction in a state such that it can be committed when the transaction ultimately commits. Note that a call to tpreturn() does not necessarily finalize an entire transaction. Also, even though the caller indicates success, if there are any outstanding replies or open connections, or if any work done within the service caused its transaction to be marked rollback-only, then a failed message is sent (that is, the recipient of the reply receives a TPESVCERR indication or a TPEV_SVCERR event). Note that if a transaction becomes rollback-only while in the service routine for any reason, rval should be set to TPFAIL. If TPSUCCESS is specified for a conversational service, a TPEV_SVCSUCC event is generated.

TPFAIL

The service has terminated unsuccessfully from an application standpoint. An error is reported to the program receiving the reply. That is, the call to get the reply fails and the recipient receives a TPSVCFAIL indication or a TPEV_SVCFAIL event. If the caller is in transaction mode, tpreturn() marks the transaction as rollback-only (note that the transaction may already be marked rollback-only). Barring any failures in processing the return, the caller's data is sent, if present. One reason for not sending the caller's data is when a transaction timeout has

occurred. In this case, the program waiting for the reply receives an error of TPETIME.

If rval does not contain one of these two values, TPFAIL is assumed.

An application-defined return code, rcode, may be sent to the program receiving the service reply. This code is sent regardless of the setting of rval as long as a reply can be successfully sent (that is, as long as the receiving call returns success or TPESVCFAIL, or receives one of the events TPEV_SVCSUCC or TPEV_SVCFAIL). In addition, for conversational services, this code can be sent only if the service routine has control of the connection when it issues tpreturn(). The value of rcode is available to the receiver in the variable tpurcode.

data points to the data portion of a reply to be sent. If data is non-NULL, it must point to a buffer previously obtained by a call to tpalloc(). If this is the same buffer passed to the service routine upon its invocation, its disposition is up to the communication resource manager; the service routine writer does not have to worry about whether it is freed or not. In fact, any attempt by the user to free this buffer fails. However, if the buffer passed to tpreturn() is not the same one with which the service is invoked, tpreturn() frees that buffer. len specifies the amount of the data buffer to be sent. If data points to a buffer that does not require a length to be specified, then len is ignored (and may be 0). If data points to a buffer that does require a length, len must not be zero.

If data is NULL, len is ignored. In this case, if a reply is expected by the program that invoked the service, a reply is sent with no data portion. If no reply is expected, tpreturn() frees data as necessary and returns sending no reply.

Currently, flags are reserved for future use and must be set to 0.

If the service is conversational, there are two cases where the caller's return code and the data portion are not transmitted:

- If the connection has already been terminated when the call is made (that is, the caller has received TPEV_DISCONIMM on the connection), this call simply ends the service routine and rolls back the current transaction, if one exists. In this case, the caller's data cannot be transmitted.
- If the caller does not have control of the connection, either TPEV_SVCFAIL or TPEV_SVCERR is sent to the originator of the connection as described above. Regardless of which event the originator receives, no data is transmitted; however, if the originator receives the TPEV_SVCFAIL event, the return code is available in the originator's tpurcode variable.

<<Arguments>>

- <<rval

  Specify either TPSUCCESS or TPFAIL.>>

- <<rcode

  Specify a return code defined in the application.>>

- <<data

  Specify the pointer to the buffer containing the reply data to be sent.>>

- <<len

  Specify the length of the buffer for data which will come.>>

- <<flags

  Set 0 (reserved for the future).>>

Return value

A service routine does not return any value to its caller, the communication resource manager dispatcher; thus, it is declared as a void. Service routines, however, are expected to terminate using `tpreturn()`. If a service routine returns without using `tpreturn()` (that is, it uses the C-language return statement or falls out of the function), the server returns a service error to the service requester. In addition, all open connections to subordinates are disconnected immediately, and any outstanding asynchronous replies are dropped. If the server was in transaction mode at the time of failure, the transaction is marked rollback-only. Note also that if `tpreturn()` is used outside a service routine (for example, by routines that are not services), it returns having no effect.

- Errors

  Since `tpreturn()` ends the service routine, any errors encountered either in handling arguments or in processing cannot be indicated to the function's caller. Such errors cause `tperrno` to be set to `TPESVCERR` for a program receiving the service's outcome via either `tpcall()` or `tpgetrply()`, and cause the event, `TPEV_SVCERR`, to be sent over the conversation to a program using `tpsend()` or `tprecv()`.

See also

`tpalloc()`, `tpcall()`, `tpconnect()`, `tpdiscon()`, `tpgetrply()`, `tprecv()`, `tpsend()`, `tpservice()`.

<<Notes on use with OpenTP1>>

1. <<Under the relevant version of the OpenTP1, the function `tpreturn()` will not terminate the service function. After calling the function `tpreturn()`, use `return()` to terminate the service function. If some processing is performed after the function `tpreturn()` is called, subsequent operation is unpredictable.>>
2. <<The behavior caused by XATMI errors encountered during OSI TP communication may be different from the behavior caused by errors encountered

conventional TCP/IP communication.>>

tpsend - Send a message in a conversational connection

Format

- ANSI C, C++

```
#include <xatmi.h>
int tpsend (int cd, char *data, long len, long flags,
            long *revent)
```

- K&R C

```
#include <xatmi.h>
int tpsend (cd, data, len, flags, revent)
int     cd;
char    *data;
long    len;
long    flags;
long    *revent;
```

Description

The function `tpsend()` is used to send data across an open connection to another program. The caller must have control of the connection. This function's first argument, `cd`, specifies the open connection over which data is sent. `cd` is a descriptor returned from either `tpconnect()` or the `TPSVCINFO` parameter passed to a conversational service.

The second argument, `data`, must point to a buffer previously allocated by `tpalloc()`. len specifies how much of the buffer to send. Note that if `data` points to a buffer of a type that does not require a length to be specified, `len` is ignored (and may be 0). If `data` points to a buffer that does require a length, `len` must not be zero. Also, `data` can be NULL in which case `len` is ignored (no application data is sent - this might be done, for instance, to grant control of the connection without transmitting any data). The type and sub-type of `data` must match one of the types and sub-types recognized by the other end of the connection.

<<Arguments>>

- <<`cd`

  Specify a descriptor.>>

- <<`data`

  Specify the pointer to the buffer containing the data to be sent.>>

- <<`len`

  Specify the length of the buffer.>>

- <<flags

  Indicates flags.>>

- <<revent

  Indicates the pointer to the long-type data about the event.>>

  The valid flags are as follows:

  TPRECVONLY

  This flag signifies that, after the caller's data is sent, the caller gives up control of the connection (that is, the caller cannot issue any more tpsend() calls). When the receiver at the other end of the connection receives the data sent by tpsend(), it also receives an event (TPEV_SENDONLY) indicating that it has control of the connection (and cannot issue more any tprecv() calls).

  TPNOBLOCK

  The data and any events are not sent if a blocking condition exists (for example, the internal buffers into which the message is transferred are full). When TPNOBLOCK is not specified and a blocking condition exists, the caller blocks until the condition subsides or a timeout occurs (either transaction or blocking timeout).

  TPNOTIME

  This flag signifies that the caller is willing to block indefinitely and wants to be immune to blocking timeouts. Transaction timeouts may still occur.

  TPSIGRSTRT

  If a signal interrupts any underlying system calls, the interrupted system call is reissued.

  If an event exists for the descriptor, cd, tpsend() fails without sending the caller's data. The event type is returned in revent. Valid events for tpsend() are as follows:

  TPEV_DISCONIMM

  Received by the subordinate of a conversation, this event indicates that the originator of the conversation has either issued an immediate disconnect on the connection via tpdiscon(), or it issued tpreturn(), tx_commit() or tx_rollback() with the connection still open. This event is also returned to the originator or subordinate when a connection is broken due to a communication error (for example, a server, machine, or network failure).

  TPEV_SVCERR

  Received by the originator of a conversation, this event indicates that the subordinate of the conversation has issued tpreturn() without having control of the conversation. In addition, tpreturn() was issued in a manner different

from that described for TPEV_SVCFAIL below.

TPEV_SVCFAIL

Received by the originator of a conversation, this event indicates that the subordinate of the conversation has issued tpreturn() without having control of the conversation. In addition, tpreturn() was issued with the TPFAIL and no data (that is, rval was set to TPFAIL and data was NULL).

Because each of these events indicates an immediate disconnection notification (that is, abortive rather than orderly), data in transit may be lost. The descriptor used for the connection is no longer valid. If the two programs were participating in the same transaction, the transaction has been marked rollback-only.

Return value

The function tpsend() returns -1 on error and sets tperrno to indicate the error condition. Upon return from tpsend() where revent is set to TPEV_SVCFAIL, the tpurcode global contains an application-defined value that was set as part of tpreturn().

■ Errors

Under the following conditions, tpsend() fails and sets tperrno to one of the following values:

Return value	Return value (numeric)	Explanation
TPEINVAL	4	Invalid arguments were given (for example, data is not a pointer to a buffer allocated for tpalloc() or the value of flags is invalid).
TPEBADDESC	2	The argument cd points to an invalid descriptor.
TPETIME	13	A timeout occurred. If the caller is in transaction mode, a transaction time-out occurred and the transaction is marked rollback_only; otherwise, a blocking time-out occurred and neither TPNOBLOCK nor TPNOTIME were specified. In either case, *data and its contents are not changed. If a transaction time-out occurred, any attempts to send or receive messages on any connections or to start a new connection fail with TPETIME until the transaction has been rolled back.
TPEEVENT	22	An event occurred. data is not sent when this error occurs. The event type is returned in revent.
TPEBLOCK	3	When tpsend() for which TPNOBLOCK was specified was called, the blocking status existed.
TPEGOTSIG	15	A signal was received, but TPSIGRSTRT was not set.

Return value	Return value (numeric)	Explanation
TPEPROTO	9	tpsend() was called in an improper context.
TPESYSTEM	12	A communication resource manager system error has occurred. The exact nature of the error is determined in a product-specific manner.
TPEOS	7	An operating system error has occurred. The exact nature of the error is determined in a product-specific manner.

See also

tpalloc(), tpconnect(), tpdiscon(), tprecv(), tpreturn().

<<Notes on use with OpenTP1>>

1. <<The TPNOBLOCK flag is invalid under the relevant version of the OpenTP1. Therefore, the error code TPEBLOCK will not be returned to tperrno. The OpenTP1 is designed so that if communication is impossible because of blocking, TPESYSTEM is returned as when communication is impossible because of network failure.>>
2. <<The TPNOTIME flag is invalid under the relevant version of the OpenTP1.>>
3. <<The TPSIGRSTRT flag is invalid. Regardless of this flag, when a signal is received, the interrupted system call is reinvoked. TPEGOTSIG will never return.>>
4. <<Under the OpenTP1, when a process encounters transaction timeout, it terminates abnormally. Therefore, TPETIME returns only when blocking timeout occurs.>>
5. <<Under the relevant version of the OpenTP1, data which requires rollback causes the return of TPESYSTEM unless otherwise specified by the X/Open. However, the rollback_only state may not come into effect even when TPESYSTEM returns.>>
6. <<Under the OpenTP1, even if the mate of conversation has called the function tpdiscon() or tpreturn(), the function tpsend() cannot generate an event provided that the process which calls the function tpsend() has not received an event.>>
7. <<For OSI TP communication using TP1/NET/OSI-TP-Extended, conversational service communication cannot be held. If this is done, the system operation is undefined.>>

tpservice - Template for service routines

Format

- ANSI C, C++

```
#include <xatmi.h>
void tpservice (TPSVCINFO *svcinfo)
```

- K&R C

```
#include <xatmi.h>
void tpservice (svcinfo)
TPSVCINFO    *svcinfo;
```

Description

The function `tpservice()` is the template for writing service routines. This template is used for services that receive requests via `tpcall()` or `tpacall()` routines as well as by services that communicate via `tpconnect()`, `tpsend()` and `tprecv()` routines.

Service routines processing requests made via either `tpcall()` or `tpacall()` receive, at most, one incoming message (in the `data` element of `svcinfo`) and send, at most, one reply (upon exiting the service routine with `tpreturn()`).

Conversational services, on the other hand, are invoked by connection requests with, at most, one incoming message along with a means of referring to the open connection. When a conversational service routine is invoked, either the connecting program or the conversational service may send and receive data as defined by the application. The connection is half-duplex in nature meaning that one side controls the conversation (that is, it sends data) until it explicitly gives up control to the other side of the connection.

Concerning transactions, service routines can participate in, at most, one transaction if invoked in transaction mode. As far as the service routine writer is concerned, the transaction ends upon returning from the service routine. If the service routine is not invoked in transaction mode, the service routine may originate as many transactions as it wants using `tx_begin()`, `tx_commit()` and `tx_rollback()`. Note that `tpreturn()` is not used to complete a transaction. Thus, it is an error to call `tpreturn()` with an outstanding transaction that originated within the service routine.

<<Argument>>

Service routines are invoked with one argument: `svcinfo`, a pointer to a service information structure. This structure includes the following members:

```
char      name[XATMI_SERVICE_NAME_LENGTH];
char      *data;
long      len;
long      flags;
int       cd;
```

The element `name` is populated with the service name that the requester used to invoke the service.

The setting of `flags` upon entry to a service routine indicates attributes that the service routine may want to note. The possible values for `flags` are as follows:

`TPCONV`

A connection request for a conversation has been accepted and the descriptor for the conversation is available in `cd`. If not set, this is a request/response service and `cd` is not valid.

`TPTRAN`

The service routine is in transaction mode.

`TPNOREPLY`

The caller is not expecting a reply. This option is not set if `TPCONV` is set.

`TPSENDONLY`

The service is invoked such that it can send data across the connection and the program on the other end of the connection can only receive data. This flag is mutually exclusive with `TPRECVONLY` and may be set only when `TPCONV` is also set.

`TPRECVONLY`

The service is invoked such that it can only receive data from the connection and the program on the other end of the connection can send data. This flag is mutually exclusive with `TPSENDONLY` and may be set only when `TPCONV` is also set.

The element data points to the data portion of a request message and `len` is the length of the `data`. The buffer pointed to by data was allocated by `tpalloc()` in the communication resource manager. This buffer may be grown by the user with `tprealloc()`; however, it cannot be freed by the user. It is recommended that this buffer be the one passed to `tpreturn()` when the service ends. If a different buffer is passed to those routines, that buffer is freed by them. Note that the buffer pointed to by `data` is overwritten by the next service request even if this buffer is not passed to `tpreturn()`. The element `data` may be NULL if no data accompanied the request. In this case, `len` is 0.

When `TPCONV` is set in `flags`, `cd` is the connection descriptor that can be used with

tpsend() and tprecv() to communicate with the program that initiated the conversation.

Return value

A service routine does not return any value to its caller, the communication resource manager dispatcher; thus, it is declared as a void. Service routines, however, are expected to terminate using tpreturn(). If a service routine returns without using tpreturn() (that is, it uses the C-language return statement or falls out of the function), the server returns a service error to the service requester. In addition, all open connections to subordinates are disconnected immediately, and any outstanding asynchronous replies are dropped. If the server was in transaction mode at the time of failure, the transaction is marked rollback-only. Note also that if tpreturn() is used outside a service routine (for example, by routines that are not services), then it returns having no effect.

- Errors

Since tpreturn() ends the service routine, any errors encountered either in handling arguments or in processing cannot be indicated to the function's caller. Such errors cause tperrno to be set to TPESVCERR for a program receiving the service's outcome via either tpcall() or tpgetrply(), and cause the event, TPEV_SVCERR, to be sent over the conversation to a program using tpsend() or tprecv().

See also

tpalloc(), tpcall(), tpconnect(), tpgetrply(), tprecv(), tpreturn(), tpsend().

<<Notes on using the function in OpenTP1>>

1. <<For an OpenTP1 UAP (service function), always write return immediately after tpreturn(). This is because OpenTP1 execution processes are restricted. After calling tpreturn(), immediately execute return. No processing must be performed between tpreturn() and return. Updating resources between a call to tpreturn() and execution of return within transaction processing includes the updating in the transaction.>>
2. <<The behavior caused by XATMI errors encountered during OSI TP communication may be different from the behavior caused by errors encountered conventional TCP/IP communication.>>

tptypes - Determine information about a typed buffer

Format

- ANSI C, C++

```
#include <xatmi.h>
long tptypes (char *ptr, char *type, char *subtype)
```

- K&R C

```
#include <xatmi.h>
long tptypes (ptr, type, subtype)
char    *ptr;
char    *type;
char    *subtype;
```

Description

The function `tptypes()` takes as its first argument a pointer to a data buffer and returns the type and subtype of that buffer in its second and third arguments, respectively. `ptr` must point to a buffer obtained from `tpalloc()`. If `type` and `subtype` are non-NULL, the function populates the character arrays to which they point with the names of the buffer's type and subtype, respectively. If the names are of their maximum length (8 for `type`, 16 for `subtype`), the character array is not null-terminated. If no subtype exists, then the array pointed to by `subtype` contains a NULL string ("").

Note that only the first eight bytes of `type` and the first 16 bytes of `subtype` are populated.

<<Arguments>>

- <<`ptr`

  Specify the pointer to the buffer.>>

- <<`type`

  Specify the pointer to the buffer `type`.>>

- <<`subtype`

  Specify the pointer to the buffer `subtype`.>>

Return value

Upon success, `tptypes()` returns the size of the buffer. Otherwise, it returns `-1` upon failure and sets `tperrno` to indicate the error condition.

■ Errors

Under the following conditions, `tptypes()` fails and sets `tperrno` to one of the following values:

Return value	Return value (numeric)	Explanation
TPEINVAL	4	Invalid arguments were given (for example, `ptr` is not a pointer to a typed buffer).
TPEPROTO	9	`tptypes()` was called in an improper context.
TPESYSTEM	12	A communication resource manager system error has occurred. The exact nature of the error is determined in a product-specific manner.
TPEOS	7	An operating system error has occurred. The exact nature of the error is determined in a product-specific manner.

See also

`tpalloc()`, `tpfree()`, `tprealloc()`.

<<Notes on use with OpenTP1>>

1. <<The behavior caused by XATMI errors encountered during OSI TP communication may be different from the behavior caused by errors encountered conventional TCP/IP communication.>>

tpunadvertise - Unadvertise a service name

Format

■ ANSI C, C++

```
#include <xatmi.h>
int   tpunadvertise (char *svcname)
```

■ K&R C

```
#include <xatmi.h>
int   tpunadvertise (svcname)
char    *svcname;
```

Description

The function `tpunadvertise()` allows a server to unadvertise a service that it offers. By default, a server's services are advertised when it is booted and they are unadvertised when it is shutdown.

The function `tpunadvertise()` removes `svcname` as an advertised service for the server. The argument `svcname` cannot be NULL or the NULL string (""). Also, `svcname` should be 15 characters or fewer. Longer names are accepted and truncated to 15 characters. Care should be taken such that truncated names do not match other service names.

<<Argument>>

■ <<`svcname`

Specify the name of the service.>>

Return value

`tpunadvertise()` returns `-1` on error and sets `tperrno` to indicate the error condition.

■ Errors

Under the following conditions, `tpunadvertise()` fails and sets `tperrno` to one of the following values:

Return value	Return value (numeric)	Explanation
`TPEINVAL`	4	The argument `svcname` is NULL or the NULL string ("").
`TPENOENT`	6	The argument `svcname` is not currently advertised by the server.

Return value	Return value (numeric)	Explanation
TPEPROTO	9	tpunadvertise() was called in an improper context.
TPESYSTEM	12	A communication resource manager system error has occurred. The exact nature of the error is determined in a product-specific manner.
TPEOS	7	An operating system error has occurred. The exact nature of the error is determined in a product-specific manner.

See also

tpadvertise().

<<Notes on use with OpenTP1>>

1. <<Suppose that load balancing is used on one node (multiserver configuration). When the function tpunadvertise() is called from one of the processes, the service becomes unavailable to all processes which undergo load balancing. When the tpadvertise() is later called to advertise the service, service requests from the processes can be accepted.>>
2. <<Suppose that load balancing (internode load balancing facility and extended internode load-balancing facility) is used on multiple nodes. When the function tpunadvertise() is called from a process on a node, the service becomes unavailable on that node. However, the servers at other nodes can accept service requests. When the function tpadvertise() is later called to advertise the service, service requests are acceptable.>>
3. <<The behavior caused by XATMI errors encountered during OSI TP communication may be different from the behavior caused by errors encountered conventional TCP/IP communication.>>

TX-interfaced application programming interface (tx_~)

This section explains the syntax of the API functions which implement the TX interface. The text in this section is quoted from *5. C Reference Manual Pages* which is the syntax reference section of the *X/Open CAE Specification Distributed TP: The TX (Transaction Demarcation) Specification* published by X/Open Company Limited.

Additional notes on using these functions from UAPs used with the OpenTP1 are enclosed in symbols <<>>.

The syntax of the following functions is explained below:

- `tx_begin` - Begin a global transaction
- `tx_close` - Close a set of resource managers
- `tx_commit` - Commit a global transaction
- `tx_info` - Return global transaction information
- `tx_open` - Open a set of resource managers
- `tx_rollback` - Roll back a global transaction
- `tx_set_commit_return` - Set `commit_return` characteristic
- `tx_set_transaction_control` - Set `transaction_control` characteristic
- `tx_set_transaction_timeout` - Set `transaction_timeout` characteristic

TX interface functions (`tx_~`) can be used in the UAPs for both TP1/Server Base and TP1/LiNK.

tx_begin - Begin a transaction

Format

- ANSI C, C++

```
#include <tx.h>
int  tx_begin (void)
```

- K&R C

```
#include <tx.h>
int  tx_begin()
```

Description

The function `tx_begin()` is used to place the calling thread of control in transaction mode. The calling thread must first ensure that its linked resource managers have been opened (by means of `tx_open()`) before it can start transactions. The function `tx_begin()` fails (returning `TX_PROTOCOL_ERROR`) if the caller is already in transaction mode or `tx_open()` has not been called.

Once in transaction mode, the calling thread must call `tx_commit()` or `tx_rollback()` to complete its current transaction. There are certain cases related to transaction chaining where `tx_begin()` does not need to be called explicitly to start a transaction. See `tx_commit()` and `tx_rollback()` for details.

<<`tx_begin()` cannot be called by MHP.>>

<<The value set by the following function affects the processing of `tx_begin()`.>>

Optional set-up

- `tx_set_transaction_timeout()`

Return value

<<When return value is 0>> upon successful completion, `tx_begin()` returns `TX_OK`, a non-negative return value.

- Errors

Under the following conditions, `tx_begin()` fails and returns one of these negative values.

Return value	Return value (numeric)	Explanation
TX_OUTSIDE	-1	The transaction manager is unable to start a global transaction because the calling thread of control is currently participating in work outside any global transaction with one or more resource managers. All such work must be completed before a global transaction can be started. The caller's status with respect to the local transaction is unchanged.
TX_PROTOCOL_ERROR	-5	The function was called in an improper context (for example, the caller is already in transaction mode). The caller's status with respect to transaction mode is unchanged.
TX_ERROR	-6	Either the transaction manager or one or more of the resource managers encountered a transient error trying to start a new transaction. When this error is returned, the caller is not in transaction mode.
TX_FAIL	-7	Either the transaction manager or one or more of the resource managers encountered a fatal error. The nature of the error is such that the transaction manager and/or one or more of the resource managers can no longer perform work on behalf of the application. When this error is returned, the caller is not in transaction mode.

Application usage

XA-compliant resource managers must be successfully opened to be included in the global transaction. (See `tx_open()`, for details.)

See also

`tx_commit()`, `tx_open()`, `tx_rollback()`, `tx_set_transaction_timeout()`.

<<Example>>

```
<<if (tx_info (NULL) == 0 && tx_begin() < 0)
    fputs ("cannot begin transaction\n", stderr);>>
```

<<Note on use with OpenTP1>>

1. <<`tx_begin()` must be called when transaction processing is started with SPP. For SPP, transaction processing is started if `tx_begin()` is called by the caller.>>

2. <<For the process that generates a transaction with `tx_begin()`, the executable file of UAP which is correctly linked according to the description of this manual must be started.>>
3. <<`tx_begin()` cannot be used along with the functions `dc_trn_~()`.>>

tx_close - Close a set of resource managers

Format

- ANSI C, C++

```
#include <tx.h>
int  tx_close (void)
```

- K&R C

```
#include <tx.h>
int  tx_close()
```

Description

The function `tx_close()` closes a set of resource managers in a portable manner. It invokes a transaction manager to read information specific to the resource manager in a manner specific to the transaction manager and pass this information to the resource managers linked to the caller.

The function `tx_close()` closes all resource managers to which the caller is linked. This function is used in place of close calls specific to the resource manager and allows an application program to be free of calls, which may hinder portability. Since resource managers differ in their termination semantics, the specific information needed to close a particular resource manager must be published by each resource manager.

The function `tx_close()` should be called when an application thread of control no longer wishes to participate in global transactions. The function `tx_close()` fails (returning `TX_PROTOCOL_ERROR`) if the caller is in transaction mode. That is, no resource managers are closed even though some may not be participating in the current transaction.

When `tx_close()` returns success (`TX_OK`), all resource managers linked to the calling thread are closed.

Return value

<<When return value is 0>>

Upon successful completion, `tx_close()` returns `TX_OK`, a non-negative return value. <<The set of resource managers linked to the caller was closed.>>

- Errors

Under the following conditions, `tx_close()` fails and returns one of these negative values.

Return value	Return value (numeric)	Explanation
TX_PROTOCOL_ERROR	-5	The function was called in an improper context (for example, the caller is in transaction mode). No resource managers are closed.
TX_ERROR	-6	Either the transaction manager or one or more of the resource managers encountered a transient error. All resource managers that could be closed are closed.
TX_FAIL	-7	Either the transaction manager or one or more of the resource managers encountered a fatal error. The nature of the error is such that the transaction manager and/or one or more of the resource managers can no longer perform work on behalf of the application.

See also

tx_open().

<<Example>>

```
<<if (tx_info (NULL) == 0 && tx_close() < 0)
    fputs ("cannot close resource manager\n", stderr);>>
```

<<Note on use with OpenTP1>>

1. <<Only the resource managers conforming to the XA interface of X/Open can be closed with tx_close().>>

tx_commit - Commit a global transaction

Format

- ANSI C, C++

```
#include <tx.h>
int  tx_commit (void)
```

- K&R C

```
#include <tx.h>
int  tx_commit()
```

Description

The function `tx_commit()` is used to commit the work of the transaction active in the caller's thread of control.

If the `transaction_control` characteristic (see `tx_set_transaction_control()`) is `TX_UNCHAINED`, when `tx_commit()` returns, the caller is no longer in transaction mode. However, if the `transaction_control` characteristic is `TX_CHAINED`, when `tx_commit()` returns, the caller remains in transaction mode on behalf of a new transaction (see the Return value and Errors sections below).

<<The values set by the following functions affect the processing of `tx_commit()`>>

Optional set-up

- `tx_set_commit_return()`
- `tx_set_transaction_control()`
- `tx_set_transaction_timeout()`

Return value

<<When return value is 0>>

Upon successful completion, `tx_commit()` returns `TX_OK`, a non-negative return value. <<If the `transaction_control` characteristic is `TX_CHAINED`, a new global transaction begins.>>

- Errors

Under the following conditions, `tx_commit()` fails and returns one of these negative values.

Return value	Return value (numeric)	Explanation
TX_NO_BEGIN	-100	The transaction committed successfully; however, a new transaction could not be started and the caller is no longer in transaction mode. This return value occurs only when the transaction characteristic is TX_CHAINED.
TX_ROLLBACK	-2	The transaction could not commit and has been rolled back. In addition, if the transaction_control characteristic is TX_CHAINED, a new transaction is started.
TX_ROLLBACK_NO_BEGIN	-102	The transaction could not commit and has been rolled back. In addition, a new transaction could not be started and the caller is no longer in transaction mode. This return value can occur only when the transaction characteristic is TX_CHAINED.
TX_MIXED	-3	The transaction was partially committed and partially rolled back. In addition, if the transaction_control characteristic is TX_CHAINED, a new transaction is started.
TX_MIXED_NO_BEGIN	-103	The transaction was partially committed and partially rolled back. In addition, a new transaction could not be started and the caller is no longer in transaction mode. This return value can occur only when the transaction characteristic is TX_CHAINED.
TX_HAZARD	-4	Due to a failure, the transaction may have been partially committed and partially rolled back. In addition, if the transaction_control characteristic is TX_CHAINED, a new transaction is started. This function returns TX_HAZARD even when you specify 00000001 for the trn_extend_function operand in the transaction service definition and the return value from the resource manager at one-phase commit is XAER_NOTA.

Return value	Return value (numeric)	Explanation
TX_HAZARD_NO_BEGIN	-104	Due to a failure, the transaction may have been partially committed and partially rolled back. In addition, a new transaction could not be started and the caller is no longer in transaction mode. This return value can occur only when the transaction characteristic is TX_CHAINED. This function returns TX_HAZARD_NO_BEGIN even when you specify 00000001 for the trn_extend_function operand in the transaction service definition and the return value from the resource manager at one-phase commit is XAER_NOTA.
TX_PROTOCOL_ERROR	-5	The function was called in an improper context (for example, the caller is not in transaction mode). The caller's state with respect to transaction is not changed.
TX_FAIL	-7	Either the transaction manager or one or more of the resource managers encountered a fatal error. The nature of the error is such that the transaction manager and/or one or more of the resource managers can no longer perform work on behalf of the application. The caller's state with respect to the transaction is unknown.

See also

tx_begin(), tx_set_commit_return(), tx_set_transaction_control(), tx_set_transaction_timeout().

<<Example>>

```
<<if (tx_info (NULL) == 1 && tx_commit() <0)
    fputs ("cannot commit transaction\n", stderr);>>
```

<<Note on use with OpenTP1>>

1. <<tx_commit() can be issued only by a process of the UAP which started the global transaction (UAP which called tx_begin()).>>
2. <<For the process that issues tx_commit(), the executable file of UAP which is correctly linked according to the description of this manual must be started.>>
3. <<tx_commit() cannot be used along with the functions dc_trn_~().>>

tx_info - Return global transaction information

Format

- ANSI C, C++

```
#include <tx.h>
int  tx_info (TXINFO *info)
```

- K&R C

```
#include <tx.h>
int  tx_info (info)
TXINFO *info
```

Description

The function `tx_info()` returns global transaction information in the structure pointed to by `info`. In addition, this function returns a value indicating whether the caller is currently in transaction mode or not.

<<Argument>>

<<info>>

If `info` is non-null, `tx_info()` populates a `TXINFO` structure pointed to by `info` with global transaction information. The `TXINFO` structure contains the following elements:

```
XID                   xid;
COMMIT_RETURN         when_return;
TRANSACTION_CONTROL   transaction_control;
TRANSACTION_TIMEOUT   transaction_timeout;
TRANSACTION_STATE     transaction_state;
```

If `tx_info()` is called in transaction mode, `xid` is populated with a current transaction branch identifier and `transaction_state` contains the state of the current transaction. If the caller is not in transaction mode, `xid` is populated with the null XID (see <tx.h> for details). In addition, regardless of whether the caller is in transaction mode, `when_return`, `transaction_control`, and `transaction_timeout` contain the current settings of the `commit_return` and `transaction_control` characteristics, and the transaction timeout value in seconds.

The transaction timeout value returned reflects the setting that is used when the next transaction is started. Thus, it may not reflect the timeout value for the caller's current global transaction since calls made to `tx_set_transaction_timeout()` after the current transaction was begun may have changed its value.

If `info` is null, no `TX_INFO` structure is returned.

Return value

If the caller is in transaction mode, 1 is returned. If the caller is not in transaction mode, 0 is returned.

- Errors

  Under the following conditions, `tx_info()` fails and returns one of these negative values.

Return value	Return value (numeric)	Explanation
`TX_PROTOCOL_ERROR`	`-5`	The function was called in an improper context (for example, the caller has not yet called `tx_open()`).
`TX_FAIL`	`-7`	The transaction manager encountered a fatal error. The nature of the error is such that the transaction manager can no longer perform work on behalf of the application.

Application usage

Within the same global transaction, subsequent calls to `tx_info()` are guaranteed to provide an XID with the same gtrid component, but not necessarily the same `bqual` component.

See also

`tx_open()`, `tx_set_commit_return()`, `tx_set_transaction_control()`, `tx_set_transaction_timeout()`.

<<Example>>

```
<<if (tx_info (NULL) !=1)
    fputs ("not transaction mode\n", stderr);>>
```

tx_open - Open a set of resource managers

Format

- ANSI C, C++

```
#include <tx.h>
int  tx_open (void)
```

- K&R C

```
#include <tx.h>
int  tx_open()
```

Description

The function `tx_open()` opens a set of resource managers in a portable manner. It invokes a transaction manager to read information specific to the resource manager in a manner specific to the transaction manager and pass this information to the resource managers linked to the caller.

The function `tx_open()` attempts to open all resource managers that have been linked with the application. This function is used in place of open calls specific to the resource manager and allows an application program to be free of calls, which may hinder portability. Since resource managers differ in their initialization semantics, the specific information needed to open a particular resource manager must be published by each resource manager.

If `tx_open()` returns `TX_ERROR`, no resource managers are open. If `tx_open()` returns `TX_OK`, some or all of the resource managers have been opened. Resource managers that are not open return errors specific to the resource manager when accessed by the application. The function `tx_open()` must successfully return before a thread of control participates in global transactions.

Once `tx_open()` returns success, subsequent calls to `tx_open()` (before an intervening call to `tx_close()`) are allowed. However, such subsequent calls return success, and the TM does not attempt to reopen any RMs.

Return value

<<When return value is 0>>

Upon successful completion, `tx_open()` returns `TX_OK`, a non-negative return value. <<The set of one or more resource managers linked to the caller was opened.>>

■ Errors

Under the following conditions, `tx_open()` fails and returns one of these negative values.

Return value	Return value (numeric)	Explanation
`TX_ERROR`	`-6`	Either the transaction manager or one or more of the resource managers encountered a transient error. No resource managers are open.
`TX_FAIL`	`-7`	Either the transaction manager or one or more of the resource managers encountered a fatal error. The nature of the error is such that the transaction manager and/or one or more of the resource managers can no longer perform work on behalf of the application. Alternatively, an error occurred in the transaction manager because the execution environment was in non-journal operation mode.

See also

`tx_close()`.

<<Example>>

```
<<if ( tx_open() <0)
    fputs ("cannot open resource manager\n", stderr);>>
```

<<Note on use with OpenTP1>>

1. <<Only the resource managers conforming to the XA interface of X/Open can be opened with `tx_open()`.>>
2. <<`tx_open()` cannot used along with the function `dc_trn_~()`.>>

tx_rollback - Roll back a global transaction

Format

- ANSI C, C++

```
#include <tx.h>
int  tx_rollback (void)
```

- K&R C

```
#include <tx.h>
int  tx_rollback()
```

Description

The function `tx_rollback()` is used to roll back the work of the transaction active in the caller's thread of control.

If the `transaction_control` characteristic (see `tx_set_transaction_control()`) is `TX_UNCHAINED`, when `tx_rollback()` returns, the caller is no longer in transaction mode. However, if the `transaction_control` characteristic is `TX_CHAINED`, when `tx_rollback()` returns, the caller remains in transaction mode on behalf of a new transaction (see the Return value and Errors sections below).

<<The values set by the following functions affect the processing of `tx_rollback()`.>>

Optional set-up

- `tx_set_transaction_control()`
- `tx_set_transaction_timeout()`

<<`tx_rollback()` cannot be called by MHP.>>

Return value

<<When return value is 0>>

Upon successful completion, `tx_rollback()` returns `TX_OK`, a non-negative return value.

<<If the `transaction_control` characteristic is `TX_CHAINED`, a new global transaction begins.>> <<If the SPP which issued `tx_rollback()` is not root transaction branch, actual rollback processing is not performed, and it is only recorded that the transaction branch is in `rollback_only` state. The transaction mode is kept until rollback is directed in the synchronization point processing of the root transaction

branch.>>

■ Errors

Under the following conditions, `tx_rollback()` fails and returns one of these negative values.

Return value	Return value (numeric)	Explanation
TX_NO_BEGIN	-100	The transaction rolled back; however, a new transaction could not be started and the caller is no longer in transaction mode. This return value occurs only when the transaction_control characteristic is TX_CHAINED.
TX_MIXED	-3	The transaction was partially committed and partially rolled back. In addition, if the transaction_control characteristic is TX_CHAINED, a new transaction is started.
TX_MIXED_NO_BEGIN	-103	The transaction was partially committed and partially rolled back. In addition, a new transaction could not be started and the caller is no longer in transaction mode. This return value can occur only when the transaction_control characteristic is TX_CHAINED.
TX_HAZARD	-4	Due to a failure, the transaction may have been partially committed and partially rolled back. In addition, if the transaction_control characteristic is TX_CHAINED, a new transaction is started.
TX_HAZARD_NO_BEGIN	-104	Due to a failure, the transaction may have been partially committed and partially rolled back. In addition, a new transaction could not be started and the caller is no longer in transaction mode. This return value can occur only when the transaction_control characteristic is TX_CHAINED.
TX_COMMITTED	-9	The transaction was heuristically committed. In addition, if the transaction_control characteristic is TX_CHAINED, a new transaction is started.
TX_COMMITTED_NO_BEGIN	-109	The transaction was heuristically committed. In addition, a new transaction could not be started and the caller is no longer in transaction mode. This return value can occur only when the transaction_control characteristic is TX_CHAINED.
TX_PROTOCOL_ERROR	-5	The function was called in an improper context (for example, the caller is not in transaction mode).

Return value	Return value (numeric)	Explanation
TX_FAIL	-7	Either the transaction manager or one or more of the resource managers encountered a fatal error. The nature of the error is such that the transaction manager and/or one or more of the resource managers can no longer perform work on behalf of the application. The caller's state with respect to the transaction is unknown.

See also

tx_begin(), tx_set_transaction_control(), tx_set_transaction_timeout()

<<Example>>

```
<<if (tx_info (NULL) == 1 && tx_rollback() < 0)
    fputs ("cannot rollback transaction\n", stderr);>>
```

<<Note on use with OpenTP1>>

1. <<When the transaction characteristic is TX_CHAINED, tx_rollback() can be called only by the root transaction branch (UAP which called tx_begin()).>>
2. <<When the transaction characteristic is TX_UNCHAINED, tx_rollback() can be called by other than the root transaction branch. In this case, processing differs depending on the transaction branch which called tx_rollback(). When the caller of tx_rollback() is the root branch, rollback request is called to non-root branches via RPC function. When tx_rollback() is called by a non-root branch, the caller only records rollback_only and does not call rollback request to the root branch via RPC function. This non-root branch performs rollback processing after waiting for the direction by the root branch.>>
3. <<tx_rollback() cannot be used along with the functions dc_trn_~().>>

tx_set_commit_return - Set commit_return characteristic

Format

- ANSI C, C++

```
#include <tx.h>
int  tx_set_commit_return (COMMIT_RETURN  when_return)
```

- K&R C

```
#include <tx.h>
int  tx_set_commit_return (when_return)
COMMIT_RETURN when_return
```

Description

The function tx_set_commit_return() sets the commit_return characteristic to the value specified in when_return. This characteristic affects the way tx_commit() behaves with respect to returning control to its caller.

tx_set_commit_return() may be called regardless of whether its caller is in transaction mode. This setting remains in effect until changed by a subsequent call to tx_set_commit_return().

The initial setting for this characteristic is implementation dependent <<in the case of OpenTP1, TX_COMMIT_COMPLETED.>>

<<Argument>>

- <<when_return>>

  The valid settings for when_return are as follows:

  {TX_COMMIT_DECISION_LOGGED|TX_COMMIT_COMPLETED}

  - TX_COMMIT_DECISION_LOGGED

    <<This argument is not supported by the corresponding version of OpenTP1. If TX_COMMIT_DECISION_LOGGED is set for when_return, error is returned with return value TX_NOT_SUPPORTED.>>

    This flag indicates that tx_commit() should return after the commit decision has been logged by the first phase of the two-phase commit protocol but before the second phase has completed. This setting allows for faster response to the caller of tx_commit(). However, there is a risk that a transaction has a heuristic outcome, in which case the caller does not find out about this situation by means

of status codes from tx_commit(). Under normal conditions, participants that promise to commit during the first phase do so during the second phase. In certain unusual circumstances however (for example, long-lasting network or node failures) phase 2 completion may not be possible and heuristic results may occur. A transaction manager may optionally choose not to support this feature and may return TX_NOT_SUPPORTED to indicate that this value is not supported.

- TX_COMMIT_COMPLETED

  This flag indicates that tx_commit() should return after the two-phase commit protocol has finished completely. This setting allows the caller of tx_commit() to see return codes that indicate that a transaction had or may have had heuristic results. A transaction manager may optionally choose not to support this feature and may return TX_NOT_SUPPORTED to indicate that this value is not supported.

Return value

<<When return value is 0>>

Upon successful completion, tx_set_commit_return() returns TX_OK, a non-negative return value.

<<When return value is positive>>

If the transaction manager does not support the setting of when_return to TX_COMMIT_DECISION_LOGGED, it returns TX_NOT_SUPPORTED, a non-negative return value, and the commit_return characteristic remains set to its existing value. The transaction manager must support the setting of when_return to at least one of TX_COMMIT_COMPLETED or TX_COMMIT_DECISION_LOGGED. <<For OpenTP1, the return value is TX_COMMIT_RETURN.>>

■ Errors

Under the following conditions, tx_set_commit_return() does not change the setting of the commit_return characteristic and returns one of these negative values.

Return value	Return value (numeric)	Explanation
TX_EINVAL	-8	The value set for when_return is neither TX_COMMIT_DECISION_LOGGED nor TX_COMMIT_COMPLETED.
TX_PROTOCOL_ERROR	-5	The function was called in an improper context (for example, the caller has not yet called tx_open()).
TX_FAIL	-7	The transaction manager encountered a fatal error. The nature of the error is such that the transaction manager can no longer perform work on behalf of the application.

See also

tx_commit(), tx_open(), tx_info().

<<Example>>

```
<<if (tx_set_commit_return (TX_COMMIT_COMPLETED) == 0 &&
                                                tx_commit() < 0 )
     fputs ("cannot commit transaction\n", stderr);>>
```

<<Note on use with OpenTP1>>

1. <<tx_set_commit_return() cannot be used along with the functions dc_trn_~().>>

tx_set_transaction_control - Set transaction_control characteristic

Format

■ ANSI C, C++

```
#include <tx.h>
int  tx_set_transaction_control (TRANSACTION_CONTROL control)
```

■ K&R C

```
#include <tx.h>
int  tx_set_transaction_control (control)
TRANSACTION_CONTROL control
```

Description

The function `tx_set_transaction_control()` sets the `transaction_control` characteristic to the value specified in `control`. This characteristic determines whether `tx_commit()` and `tx_rollback()` start a new transaction before returning to their caller.

The function `tx_set_transaction_control()` may be called regardless of whether the application program is in transaction mode. This setting remains in effect until changed by a subsequent call to `tx_set_transaction_control()`.

The initial setting for this characteristic is `TX_UNCHAINED`.

<<Argument>>

■ <<control>>

The valid settings for `control` are as follows:

{TX_UNCHAINED|TX_CHAINED}

- `TX_UNCHAINED`

  This flag indicates that `tx_commit()` and `tx_rollback()` should not start a new transaction before returning to their caller. The caller must issue `tx_begin()` to start a new transaction.

- `TX_CHAINED`

  This flag indicates that `tx_commit()` and `tx_rollback()` should start a new transaction before returning to their caller.

Return value

<<When return value is 0>>

Upon successful completion, tx_set_transaction_control() returns TX_OK, a non-negative return value. <<The transaction_control characteristic was set to the value of control.>>

- Errors

Under the following conditions, tx_set_transaction_control() does not change the setting of the transaction_control characteristic and returns one of these negative values.

Return value	Return value (numeric)	Explanation
TX_EINVAL	-8	The value set for control is neither TX_UNCHAINED nor TX_UNCHAINED.
TX_PROTOCOL_ERROR	-5	The function was called in an improper context (for example, the caller has not yet called tx_open()).
TX_FAIL	-7	The transaction manager encountered a fatal error. The nature of the error is such that the transaction manager can no longer perform work on behalf of the application.

See also

tx_begin(), tx_commit(), tx_open(), tx_rollback(), tx_info().

<<Example>>

```
<<if (tx_set_transaction_return (TX_UNCHAINED) == 0 &&
                                          tx_commit() < 0 )
  fputs ("cannot commit transaction\n", stderr);>>
```

<<Note on use with OpenTP1>>

1. <<tx_set_transaction_control() cannot be used along with the functions dc_trn_~().>>

tx_set_transaction_timeout - Set transaction_timeout characteristic

Format

- ANSI C, C++

```
#include <tx.h>
int  tx_set_transaction_timeout (TRANSACTION_TIMEOUT
                                 timeout)
```

- K&R C

```
#include <tx.h>
int tx_set_transaction_timeout (timeout)
TRANSACTION_TIMEOUT timeout
```

Description

The function `tx_set_transaction_timeout()` sets the `transaction_timeout` characteristic to the value specified in `timeout`. This value specifies the time period in which the transaction must complete before becoming susceptible to transaction timeout; that is, the interval between the AP calling `tx_begin()` and `tx_commit()` or `tx_rollback()`.

The function `tx_set_transaction_timeout()` may be called regardless of whether its caller is in transaction mode or not. If `tx_set_transaction_timeout()` is called in transaction mode, the new timeout value does not take effect until the next transaction.

The initial `transaction_timeout` value is 0 (no timeout).

<<Argument>>

- <<`timeout`

The argument `timeout` specifies the number of seconds allowed before the transaction becomes susceptible to transaction timeout. It may be set to any value up to the maximum value for a type long as defined by the system. A `timeout` value of zero disables the timeout feature.>>

Return value

<<When return value is 0>>

Upon successful completion, `tx_set_transaction_timeout()` returns `TX_OK`, a non-negative return value. <<The `transaction_timeout` characteristic is the value set for timeout.>>

■ Errors

Under the following conditions, `tx_set_transaction_timeout()` does not change the setting of the `transaction_timeout` characteristic and returns one of these negative values.

Return value	Return value (numeric)	Explanation
`TX_EINVAL`	`-8`	The `timeout` value specified is invalid.
`TX_PROTOCOL_ERROR`	`-5`	The function was called in an improper context (for example, the caller has not yet called `tx_open()`).
`TX_FAIL`	`-7`	The transaction manager encountered an error. The nature of the error is such that the transaction manager can no longer perform work on behalf of the application.

See also

`tx_begin()`, `tx_commit()`, `tx_open()`, `tx_rollback()`, `tx_info()`.

<<Example>>

```
<<if (tx_set_transaction_timeout (TRANSACTION_TIMEOUT)
                                        == 0 && tx_commit() < 0 )
    fputs ("cannot commit transaction\n", stderr);>>
```

<<Note on use with OpenTP1>>

1. <<`tx_set_transaction_timeout()` cannot be used along with the functions `dc_trn_~()`.>>

Chapter

5. Syntax of OpenTP1 Library Functions (Association Status Notification)

Client/server communication using OSI TP as the communication protocol requires SPPs for a communication event. This chapter explains the library functions used by SPPs for a communication event and the formats of receive communication events.

This chapter contains the following sections:

Association operation (dc_xat_~)

This section explains the association operation functions used by SPPs for a communication event. An association operation function is as follows:

- `dc_xat_connect` - Establish an association

Association operation functions (`dc_xat_~`) can be used only for TP1/Server Base. For TP1/LiNK, no association operation function can be used.

Only the SPPs for a communication event can call association operation functions. The other OpenTP1 UAPs (SUP, SPP, and MHP) cannot use association operation functions.

Always specify `betran` in the `server_type` operand of the user service definition of SPPs for a communication event.

dc_xat_connect - Establish an association

Format

- ANSI C, C++

```
#include <dcxat.h>
int dc_xat_connect (char *svcname, char *aso_name,
                    DCLONG flags)
```

- K&R C

```
#include <dcxat.h>
int dc_xat_connect (svcname, aso_name, flags)
char     *svcname;
char     *aso_name:
DCLONG   flags;
```

Description

The function `dc_xat_connect()` requests the XATMI communication service specified in `svcname` to establish the association specified in `aso_name`.

The function `dc_xat_connect()` sends a request to establish an association to the remote system. Then, control is returned. The function cannot receive a report of association establishment.

The function `dc_xat_connect()` can be used only for OSI TP communication using TP1/NET/OSI-TP-Extended.

The function `dc_xat_connect()` can be called from within or outside transaction processing.

Arguments whose values are set in the UAP

- `svcname`

  Specify the name of the XATMI communication service to be requested to establish an association. As an XATMI communication service name, specify the XATMI communication service definition file name to be specified in the `xat_invoke_server` operand of the XATMI communication service definition.

- `aso_name`

  Specify the name of the association to be established. As an association name, specify the connection name specified in the `-c` option of the `nettalccn` operand of the protocol specific definition (TP1/NET/OSI-TP-Extended definition).

- flags

  Specify DCNOFLAGS.

Return values

Return value	Return value (numeric)	Meaning
DC_OK	0	Normal termination.
DCXATER_INVAL	-4570	An incorrect value is specified as the argument.
DCXATER_MEMORY	-4571	The memory became insufficient.
DCXATER_PROTO	-4572	The function dc_rpc_open() is not called.
DCXATER_NOT_FOUND	-4575	The XATMI communication service address information cannot be obtained.
DCXATER_TERMINATING	-4576	The XATMI communication service is terminating.
DCXATER_COMM_SEND	-4577	The service request failed while it was being sent to the XATMI communication service.
DCXATER_COMM_RECV	-4578	The service request failed while it was being received from the XATMI communication service. The XATMI communication service may be making a request to establish a connection.
DCXATER_ASO_NAME	-4580	The specified association name is not defined.
DCXATER_ASO_CONNECT_ALREADY	-4581	The association has already been established.
DCXATER_ASO_CONNECTING	-4582	The association is being established.
DCXATER_ASO_DISCONNECTING	-4583	The association is being released.
DCXATER_ASO_INITIATE	-4584	The association cannot be established due to the recipient mode.

Formats of receive communication events

This section explains the formats of the communication events indicating association statuses. Before receiving a communication event, specify the service group name and service name of an SPP for a communication event in the XATMI communication service definition. At this time, a receivable communication event depends on in which operands the service group name and the service name are specified.

`xat_aso_con_event_svcname` operand:

Communication event for a report of association establishment

`xat_aso_discon_event_svcname` operand:

Communication event for normal association releasing

`xat_aso_failure_event_svcname` operand:

Communication event for abnormal association releasing

If the same service group name and service name are specified in more than one operand, one SPP for a communication event can receive more than one communication event.

A communication event is reported as a structure. The structure of a communication event is defined in the header file `<dcxat.h>`. For a communication event processing SPP, include `<dcxat.h>` using `#include`.

```
struct dc_xat_event_type {
     int     event_code;       ... Communication event identification code
     char    aso_name[9];      ... Association name
     char    reserve1[3];      ... Reserved area 1
     int     aso_initiative;   ... Type of association initiating and recipient
     DCULONG reason_code;      ... Reason code
     char    xatc_svcname[9];  ... XATMI communication service name
     char    reserve2[63];     ... Reserved area 2
};
```

Contents of arguments

- `event_code`

  `event_code` contains the code identifying a communication event. The number in parentheses indicates the decimal number for an applicable code.

  `DCXAT_ASO_CONNECT` (00000001):

  Association establishment

  `DCXAT_ASO_DISCONNECT` (00000002):

  Normal association releasing

DCXAT_ASO_FAILURE (00000003):

Abnormal association releasing

- aso_name

  aso_name contains the name of the association whose status is reported by a communication event.

- reserve1

  Reserved area.

- aso_initiative

  aso_initiative contains the value indicating whether the local system is initiating or recipient through the established association. The number in parentheses indicates the decimal number for an applicable code.

  DCXAT_ASO_INIT (00000001):

  The local system is initiating side.

  DCXAT_ASO_RESP (00000002):

  The local system is recipient side.

- reason_code

  reason_code contains a reason code if an association is released. The number in parentheses indicates the decimal number for an applicable code.

  For the normal releasing of an association, reason_code contains one of the following values:

  DCXAT_RSN_COMMAND (00000001):

  Releasing of an association by executing a command.

  DCXAT_RSN_XATMI (00000005):

  Releasing of an association by the XATMI

  DCXAT_RSN_REMOTE (00000007):

  Normal releasing of an association from the remote system

  DCXAT_RSN_TP_NORMAL (00000008):

  Normal releasing of an association by the TP layer.

  For the abnormal releasing of an association, reason_code contains one of the following values:

  DCXAT_RSN_COMMAND (00000001):

  Forced releasing of an association by executing a command

DCXAT_RSN_LOWER (00000003):

Failure in a lower layer (such as a line failure and communication management failure)

DCXAT_RSN_XATMI (00000005):

Forced releasing of an association by an XATMI communication service

DCXAT_RSN_FAILURE (00000006):

Failure in association establishment

DCXAT_RSN_REMOTE (00000007):

Forced releasing of an association from the remote system

- xatc_svcname

  xatc_svcname contains an XATMI communication service name.

- reserve2

  Reserved area.

Chapter

6. X/Open-compliant Inter-application Communication (TxRPC)

This chapter explains the syntax of an interface definition language file (IDL file) and IDL compiler (`txidl` command) used in Inter-Application communication (TxRPC) defined with the X/Open.

This chapter contains the following sections:

6.1 Preparation procedures for TxRPC communication

This section explains the preparation procedures for TxRPC communication.

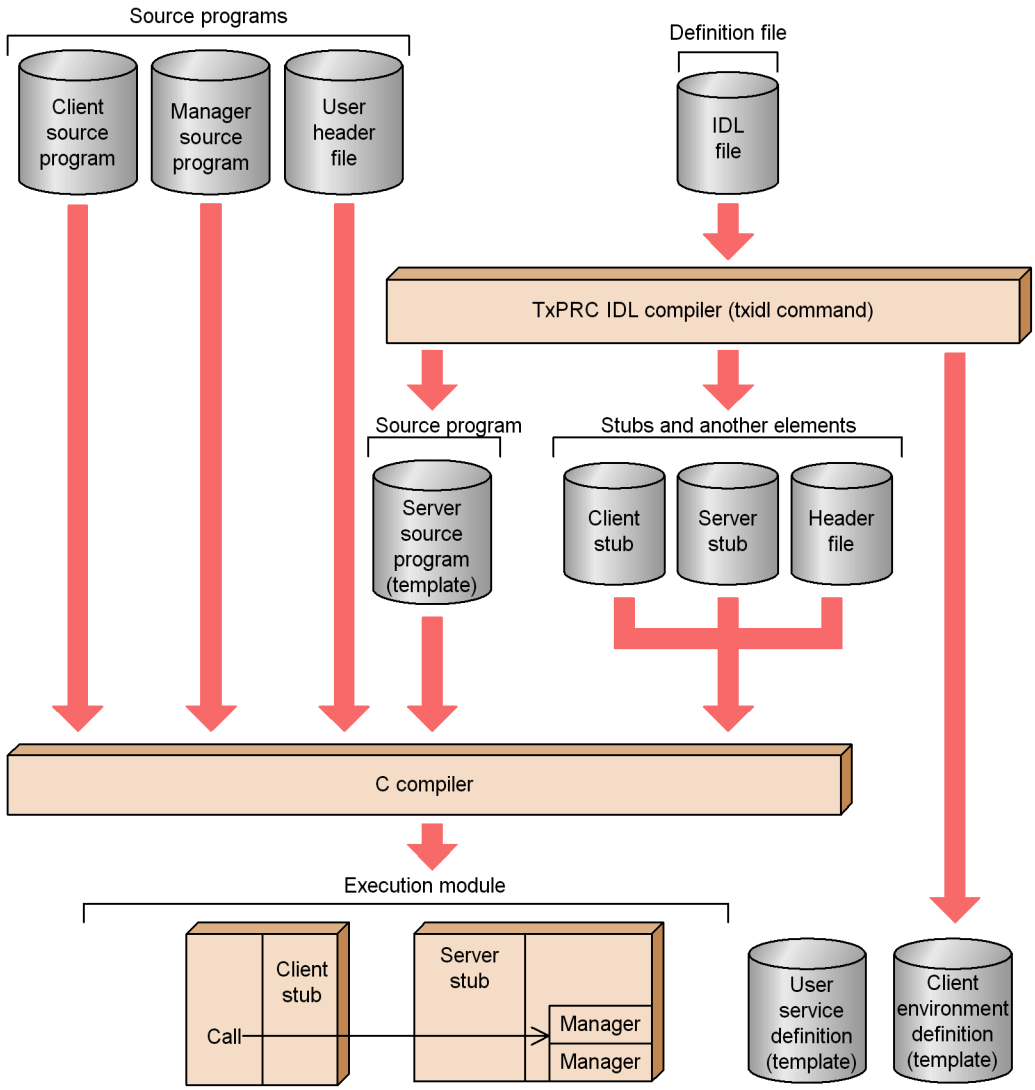
6.1.1 Procedures for using IDL-only TxRPC

To create a UAP for IDL-only TxRPC communication:

1. Create an interface definition language file (IDL file).
2. Compile the IDL file with an IDL compiler (`txidl` command).
3. Code the program based on the template of a server UAP created by the `txidl` command. Also code the client UAP.
4. Compile and link the created stub and coded program by using the `txidl` command with the C compiler.

The figure below shows the procedures for creating a UAP that communicates with IDL-only TxRPC.

Figure 6-1: Procedures for creating a UAP that communicates with IDL-only TxRPC



(1) Files created by the user

The user creates the following files:

- Client program

  Client UAP program

- Manager program

  A manager program includes operation functions corresponding to OpenTP1 service functions. This program processes requested services.

- User header file

  A user header file creates header files used in client and manager programs.

- Interface definition language file (IDL file)

  A user definition language file defines communication interfaces.

(2) Files created by the IDL compiler

The IDL compiler (`txidl` command) creates the following files:

- Client stub

  A client stub links with the client program.

- Server stub

  A server stub links with the server program.

- Header file

  A header file defines declarations for `TxRPC`.

- Template of a server program

  Template of a server program that executes user work.

- Template of a user service definition

  Template of a user service definition set for the program created by the user.

- Template of a client environment definition

  Template of a TP1/Client client environment definition set for the program created by the user. This template is created when the option to create a gateway program in the `txidl` command is specified.

Among the files noted above, the user can modify the templates of a server program, user service definition, and client environment definition before using them. For details on templates, see *7.5 TxRPC examples (templates created by the IDL compiler)*.

6.2 Notes on creating application programs

This section explains the notes on coding a UAP that communicates with TxRPC.

(1) Notes about naming the programs used in TxRPC communication

A name beginning with an alphabetic character can be arbitrarily set for an operation function (service function) except the following:

- A name beginning with `dc`
- A name beginning with `CBLDC`
- A name beginning with `tx` or `TX`
- A name beginning with `tp` or `TP`

Restrictions on other names (external variable and constant names) are the same as a UAP using the OpenTP1 library. For details on name restrictions, see *1.1.2 Coding rules*.

The above names cannot be used for coding programs and header file identifiers.

(2) Program names that cannot be used in other than TxRPC communication

With TxRPC, the interface name is used in OpenTP1. The interface name cannot be used as the service group name in another program processing.

Example:

Do not use `timope` as the service group name if the interface name is `timope`.

(3) TxRPC restrictions

The following restrictions apply to TxRPC communication:

1. With IDL-only TxRPC, the `dc_rpc_open()` and `dc_adm_complete()` functions must be called with a UAP.
2. Context handles cannot be used.
3. The macro variable cannot be declared with `#ifdef` in the IDL file.
4. A compiling error occurs with a created stub, depending on the C compiler specifications.
5. The `txidl` command does not check whether the file contents are compliant with the ANSI specifications. If an IDL file containing a description effective only with the ANSI specifications is compiled, the created stub can use only the C compiler compliant with the ANSI specifications.
6. Compile the UAP and stub with the same C compiler.

6.3 Creating interface definition language files (IDL files)

This section explains how to create an interface definition language file (IDL file).

6.3.1 Syntax rules

An IDL file must conform to the rules given below.

(1) File name

Suffix the file name with `.idl`. The same IDL file must be incorporated in the client and server UAPs.

(2) Phrase elements

(a) Identifier

Use the following characters for an identifier:

- Alphabetic characters (uppercase and lowercase)
- Numbers 0 to 9
- Underscore (_)

The first character must be an alphabetic character. Up to 31 characters can be used unless otherwise specified.

(b) Unusable term (keyword)

Some identifiers are reserved as keywords in IDL file coding. These keywords cannot be changed.

(c) Punctuation character

The following graphic characters can be used:

`", ', (, ), *, ,, . /, :, ;, |, =, [, \, ], {, }`

(d) Whitespace

The following characters are treated as whitespace:

- Spaces
- Line feeds
- Horizontal tabs
- Form feeds at the beginning of lines
- Comment lines
- Succession of one or more of the above whitespace characters

A whitespace is required at the following locations:

- Before a keyword, identifier, or number which is not prefixed with a punctuation character
- After a keyword, identifier, or number which is not suffixed with a punctuation character
- Before or/and after a punctuation character unless otherwise specified

A whitespace enclosed in double quotation marks (`"`) or single quotation marks (`'`) is treated as a character. Otherwise, whitespace is ignored because it is assumed as a delimiter of other punctuation elements.

(e) Comment

`/*` indicates the beginning and `*/` indicates the ending of a comment. Comments cannot be nested.

(3) Syntax format

The following character styles are used in this manual for explaining the syntax of coding an IDL file:

`abc`: Non-italic characters indicate that coding is to be done as noted in the expression of the syntax explanations.

abc: Italic characters indicate that coding is to be done with special values assigned.

For assignment character strings, see the syntax explanations.

The following brackets are used in this manual for explaining the syntax of coding an IDL file:

[ ]: Non-italic brackets indicate that the item must be selected. When coding an item, [ ] is needed.

[ ]: Italic brackets indicate that the item can be omitted. When coding an item, do not include *[ ]*.

6.3.2 Interface definition format

This subsection explains the format of an interface definition language (IDL) used in an IDL file. An interface definition consists of the following:

- Interface definition header

  An interface definition header defines entire interface specifications.

- Interface definition body

  An interface definition body defines individual type and operation specifications. The interface definition body consists of the following four declarations:

Import declaration

Constant declaration

Type declaration

Operation declaration

A parameter declaration is included in the operation declaration.

The declaration in the interface definition body is validated if specifications in the interface definition header and interface definition body are inconsistent.

Interface definition header

```
[[interface_attribute,...]] interface interface_name
```

Interface definition body

```
{
  import-declaration
  constant-declaration
  type-declaration
  operation-declaration
  parameter-declaration
}
```

6.3.3 Syntax of interface definition file

This subsection explains the syntax of an interface definition file in the following format:

Format

Indicates the format of each declaration in OpenTP1 IDL-only TxRPC interface definition header and body.

Meaning

Indicates the meaning of each declaration in the OpenTP1 IDL-only TxRPC interface definition header and body.

Specification item

Indicates an attribute, data type, and declarator to be specified in items given in the format. For details on attributes, see *6.6 Attributes*. For details on data types, see *6.7 Data types*. For type declarators, see *6.8 Type declarators*.

Explanation

Explains about the declarators.

OpenTP1 IDL-only TxRPC restriction

Indicates the difference between the specifications of OpenTP1 IDL-only TxRPC and IDL-only TxRPC defined with X/Open.

6.4 Syntax of interface definition header

The definition format of an interface definition header is explained.

Interface definition header

Format

```
[ [interface_attribute,...] ] interface interface_name
```

Meaning

The interface definition header defines the interface name and its attributes.

Specification item

■ *interface_attribute*

Defines interface attributes. The following attribute values can be specified:

- `version`:

  Specifies the interface version.

- `pointer_default`:

  Specifies the pointer semantics of the default value.

- `transaction_mandatory`:

  Specifies that the transaction must be expanded.

- `transaction_optional`:

  Specifies that the transaction is expanded for transaction processing.

The `transaction_mandatory` and `transaction_optional` attributes cannot be specified at the same time. Specify only one of them.

OpenTP1 IDL-only TxRPC restrictions

- Only one interface can be defined for one server.
- Specifying the `uuid` attribute is unnecessary. No error occurs even if the `uuid` attribute is specified. An error occurs, however, if the `uuid` attribute format is not conformed.
- The `local` attribute cannot be used. If used, an error occurs.
- The `endpoint` attribute cannot be used. If used, an error occurs.

- The `transaction_mandatory` and `transaction_optional` attributes are valid only if communicating processes are both `ndce` processes.

6.5 Interface definition body

The interface definition body defines one or some of the following declarations:

- Import declaration
- Constant declaration
- Type declaration
- Operation declaration (parameter declaration included)

Suffix a semicolon (;) at the end of each declaration. Enclose the interface definition body in braces { }.

Define the import declaration before other declarations. The sequence of other declarations is undefined as long as the type and constant are defined.

Import declaration

Format

```
import file,...;
```

Meaning

The import declaration imports (fetches) an interface definition file in which the type and constant to be used have been declared.

Specification item

■ *file*

Specify a file name. Enclose the name of the IDL file to be imported in double quotation marks (").

The import file name can be defined by referencing the parent directory with the -I option of the `txidl` compiler.

Explanation

1. The operation declaration is not imported.
2. The result is the same regardless how many times the interface is imported.
3. The file to be imported must be compiled with the `txidl` command in advance (only creating the header file will do).

Example

```
import "garlic.idl", "oil.idl";
```

OpenTP1 IDL-only TxRPC restriction

- Up to 100 files can be imported.

Constant declaration

Format

```
const  integer_type_spec  identifier=integer|value;
const  boolean  identifier=TRUE|FALSE|value;
const  char  identifier=character|value;
const  char*  identifier=string|value;
const  void*  identifier=NULL|value;
```

Meaning

The constant declaration declares a constant.

Specification item

The following integer constant data types can be declared:

- `integer_type_spec`: Integer constant (`hyper` excluded)
- `boolean`: Boolean constant
- `char`: Character constant
- `char*`: Character-type constant
- `void*`: Null constant

■ `identifier`

Specify a constant name.

■ `integer`, `character`, `string`, and `value`

Specify values to be allocated to the constant. So long as it is predefined, any value can be specified for the value.

Explanation

1. Do not specify `hyper`.
2. Since the constant declaration is defined with `#define` in the stub, it is expanded if the constant is used in a UAP.

Example

```
const short TEN = 10;
const boolean FAUX = FALSE;
const char CHAR = 'A';
const char* DSCH = "abcde";
```

OpenTP1 IDL-only TxRPC restriction

- A numeric expression cannot be specified as an integer constant.
- Overflow is not checked. If a value with an inappropriate size is specified, the operation is undefined.

Type declaration

Format

```
typedef  [ [type_attribute, ...] ] type_specifier  type_declarator, ...;
```

Meaning

The type declaration defines a type used by an interface.

Specification item

- *type_attribute*

  Specify the attributes of the type to be declared. The following attributes can be specified:

  - `string`

    Character string

  - `ptr`

    Complete pointer

  - `ref`

    Reference pointer

- *type_specifier*

  Specify a data type. The basic type, configuration type (structure only), or any predefined type can be specified.

- *type_declarator*

  Specify the declarator of the type to be defined. The following can be specified:

  - Simple declarator

- Fixed-length one-dimensional array
- Pointer

Explanation

1. The `string` attribute can be specified in `char` and `byte` arrays only.
2. The `ptr` and `ref` attributes can be specified only for pointers to the basic type and structure type.

OpenTP1 IDL-only TxRPC restriction

- `union` and `enum` cannot be used as the configuration type.
- A pointer to a function or array cannot be specified as a declarator.
- Adjustable and variable-length arrays cannot be used.
- Multi-dimensional arrays cannot be used.
- The following type attributes cannot be used:

  `transmit_as`, `handle`, `context_handle`, `vl_struct`, `vl_array`, `vl_string`, and `vl_enum`
- Only one pointer can be specified.
- No pointer can be specified for a structure member.
- The structure cannot be specified as a structure member.
- If the `string` attribute is specified, it is simply ignored without causing an error.

Operation declaration

Format

```
[ [operation_attribute, ...] ] type_specifier
            operation_identifier (parameter_declaration, ...);
[ [operation_attribute, ...] ] type_specifier
            operation_identifier ( [void] );
```

Meaning

The operation declaration defines a function for actual processing.

Specification item

■ *operation_attribute*

Specify an operation attribute. The following attribute can be specified:

- `transaction_mandatory`

Indicates that the transaction must be expanded.

- `transaction_optional`

  Indicates that the transaction is expanded for transaction processing.

■ *type_specifier*

Specify a data type. If a data type is returned from the operation, specify that data type. Specify a scalar type or predefined type. If no result is returned, specify `void`. The permitted type is integer.

■ *operation_identifier*

Specify an operation name. Up to 30 characters can be specified.

■ *parameter_declaration*

Specify a parameter declaration. It declares an operation parameter.

Explanation

1. The `transaction_mandatory` and `transaction_operation` attributes cannot be specified at the same time.
2. Use a complete pointer for a value returned from the operation.

OpenTP1 IDL-only TxRPC restriction

- The `context_handle` attribute cannot be used.
- The `ptr` attribute cannot be used.
- The `string` attribute cannot be used.
- The `transaction_mandatory` and `transaction_optional` attributes are valid only if the communicating processes are both `ndce` processes.
- Only `error_status_t` can be used for `type_specifier` with the corresponding version. If a system or stub error occurs, its error code is returned. The return value of the operation function is returned only when the operation terminates normally. Do not specify the pointer or array for `error_status_t`.

Parameter declaration

Format

```
[parameter_attribute, ...]  type_specifier  parameter_declarator;
```

Meaning

The parameter declaration defines operation parameters.

Specification item

- *parameter_attribute*

  Specify a parameter attribute. The following attributes can be specified:

  - `in`

    Specifies an input parameter.

  - `out`

    Specifies an output parameter.

  - `ptr`

    Specifies a complete parameter.

  - `ref`

    Specifies a reference parameter.

  - `string`

    Specifies a character string.

- *type_specifier*

  Specify a parameter data type. The following types can be specified:

  - Basic type and structure

- *parameter_declarator*

  Specify a parameter declarator. The following values can be specified:

  - Simple declarator
  - Pointer
  - Fixed-length one-dimensional array

Explanation

1. Either `in` or `out` must be specified.
2. The parameter of the `out` attribute must be an array or an explicitly declared pointer. An explicitly declared pointer is a pointer declared with *.

OpenTP1 IDL-only TxRPC restriction

- `union` and `enum` cannot be used as the configuration type.
- A pointer to a function or array cannot be specified as a declarator.
- Adjustable and variable-length arrays cannot be used.
- Multi-dimensional arrays cannot be used.

- The following type attributes cannot be used:

  Array attribute, `context_handle`, `vl_struct`, `vl_array`, `vl_string`, and `vl_enum`
- If the `string` attribute is specified, it is simply ignored without causing an error.

6.6 Attributes

This subsection explains attributes used for IDL file declaration. The following attributes can be used with OpenTP1 TxRPC:

- `version` attribute
- `pointer_default` attribute
- `transaction_mandatory` attribute
- `transaction_optional` attribute
- `in` attribute
- `out` attribute
- Pointer attribute

OpenTP1 IDL-only TxRPC restriction

- The uuid attribute is ignored with IDL-only TxRPC.
- The following attributes cause an error with IDL-only TxRPC:

  `endpoint`, `local`, `context_handle`, `transmit_as`, `vl_array`, `vl_enum`, `vl_string`, `vl_struct`, array attribute
- The `transaction_mandatory` and `transaction_optional` attributes are valid only if the communicating processes are both `ndce` processes.

The explanation formats are as follows:

Format

Indicates the format of the attribute.

Attribute meaning

Indicates the meanings of attributes.

Specification item

Indicates the items to be specified as attributes.

Explanation

Explains about attributes.

Specification example

Gives examples of attribute specification.

version attribute

Format

```
version (major [.minor])
```

Attribute meaning

The `version` attribute specifies a specific version of a remote interface.

Specification item

- *major*

  Specify this item with an integer between 0 and 65535.

- *minor*

  Specify this item with an integer between 0 and 65535.

Explanation

- Specify a version number with a set of integers indicating main version and sub-version numbers or an integer indicating the main version number only. Delimit the main version and sub-version numbers with a period (.) without inserting a space. If no sub-version is specified, 0 is assumed.
- If the `version` attribute is not specified, 0.0 is set as default.
- The client and server can communicate under the following conditions:
  - The version number of the interface called by the client is the same as the interface advertised by the server.
  - The sub-version number of the interface called by the client is the same as or lower than the interface advertised by the server.

Specification example

```
version(1.1)
version(3)
```

pointer_default attribute

Format

```
pointer_default (pointer_attribute)
```

Attribute meaning

The `pointer_default` attribute specifies which of the two pointer semantics usable in the IDL is set as the default.

Specification item

- `pointer_attribute`

  Specify either of the following pointer attributes:

  `ref`

  Reference pointer

  `ptr`

  Complete pointer

Explanation

- The default pointer semantics is used for the following pointers:
  - Pointer used for structure member declaration
  - Pointer used for other than the top level operation parameter declared by multiple pointer operators
- The pointer returned from the operation is always a complete pointer. Therefore, the `pointer_default` attribute is not used.
- The pointer attribute has the priority over the `pointer_default` attribute.
- A compiler error occurs if declaration for which a default pointer semantics is required is defined without the `pointer_default` attribute specified in the interface definition.

transaction_mandatory attribute

Format

```
transaction_mandatory
```

Attribute meaning

The `transaction_mandatory` attribute specifies that a service is to be executed as part of a global transaction.

Explanation

- The interface or operation with this attribute specified must be called inside a global transaction. If it is called outside the transaction, an error occurs and the

service is not executed.

- This attribute cannot be specified at the same time as the `transaction_optional` attribute.

transaction_optional attribute

Format

```
transaction_optional
```

Attribute meaning

The `transaction_optional` attribute specifies whether to execute a service as part of a global transaction, depending on whether the called environment is located inside or outside the transaction.

Explanation

- If the interface or operation with this attribute specified is called inside a global transaction, the service is executed as part of the transaction. If it is called outside the transaction, the service is executed as a non-transaction RPC.
- This attribute cannot be specified at the same time as the `transaction_mandatory` attribute.

in attribute

Format

```
in
```

Attribute meaning

The `in` attribute specifies that the parameter is input.

Explanation

- Either the `in` or `out` attribute must be specified for the parameter.

out attribute

Format

```
out
```

Attribute meaning

The `out` attribute specifies that the parameter is output.

Explanation

- Either the `in` or `out` attribute must be specified for the parameter.

Pointer attribute

Format

```
ref
ptr
```

Attribute meaning

The pointer attribute specifies a pointer class: reference pointer (`ref`) or complete pointer (`ptr`).

Explanation

- The pointer attribute is used for the parameter, structure member, and type definition. The `txidl` command may determine the appropriate pointer class based on how the pointer is used. In most cases, however, the pointer class needs to be specified in either of the following methods:
  1. Use the `ref` or `ptr` attribute in the pointer declaration.
  2. Use the `pointer_default` attribute for the IDL interface header. The default pointer class is determined based on the `pointer_default` attribute.
- The pointer attribute is valid only for the top level pointer in the declaration. If multiple pointers are declared in one declaration, the established `pointer_default` is validated for all pointers other than the top level pointer.
- The `ref` and `ptr` attributes cannot be specified at the same time.

Explanations of the reference pointer

A reference pointer is a simple-format pointer. The general use of the pointer is to deliver integers with reference.

The reference pointer has higher efficiency than the complete pointer; however, it has the following restrictions:

1. Linkage cannot be terminated since the reference pointer does not support NULL values.
2. A list with linkage cannot be created with the reference pointer.

The reference pointer has the following characteristics:

- The reference pointer always points to valid storage. It does not support NULL values. If a NULL value is used for the reference pointer, the operation is undefined.
- A reference pointer value is not changed during a function call. When control returns from a call, the pointer always points to the same area as at the start of the calling.
- No alias can be used. The area used by the same operation parameter and that pointed by another pointer cannot be pointed to.

Explanations of the complete pointer

A complete pointer is a complex-format pointer. The complete pointer can use all pointer-related facilities. For example, complex data structures such as a list with linkage, tree, queue, or arbitrary graph can be created.

The complete pointer has the following characteristics:

- A complete pointer value can be changed during a function call.
- No alias can be used with IDL-only TxRPC.
- The storage area for another complete pointer used by the same operation parameter can be pointed to. In this case, however, the pointer needs to point to the start of the structure. For example, the pointer to the basic structure or duplicated storage area cannot be used if the next code is incorporated in the interface definition code.

6.7 Data types

This subsection explains data types used for IDL file declaration. TxRPC data types that can be used with the OpenTP1 are as follows:

- Integer type (basic data type)
- Floating-point type (basic data type)
- Character type (basic data type)
- Boolean type (basic data type)
- Byte type (basic data type)
- `void` type (basic data type)
- Error status type (basic data type)
- Multi-language type (basic data type)
- Structure (configuration data type)

OpenTP1 IDL-only TxRPC restriction

- If the `string` attribute is specified, it is simply ignored without causing an error.
- The pointer cannot be specified as a structure member.
- The structure cannot be specified as a structure member.
- Adjustable and variable-length arrays cannot be specified.
- Multi-dimensional arrays cannot be used.
- `union` and `enum` cannot be used.
- The handle type cannot be used.

The explanation format is as follows:

Format

Indicates the data type format.

Data type explanation

Explains the data type.

Integer type (basic data type)

Format

```
int_size [ int ]
unsigned int_size [ int ]
int_size unsigned [ int ]
```

Data type explanation

The following values can be set for the `int_size`:

- `hyper` (64 bits)
- `long` (32 bits)
- `short` (16 bits)
- `small` (8 bits)

The keyword `int` is optional and has no meaning. The keyword `unsigned` indicates an unsigned integer type; it can be set before or after a size keyword.

Floating-point type (basic data type)

Format

```
float
double
```

Data type explanation

Two floating-point data lengths are available: `float`, which is 32 bits, and `double`, which is 64 bits.

Character type (basic data type)

Format

```
[ unsigned ]  char
```

Data type explanation

The keyword `unsigned` is optional and has no meaning. A signed character type cannot be used. To write a signed eight-bit integer, use the small data type.

Boolean type (basic data type)

Format

```
boolean
```

Data type explanation

The Boolean data type is expressed with eight bits. Zeros designate False, and non-zero values designate True.

Byte type (basic data type)

Format

```
byte
```

Data type explanation

- The byte `type` is expressed with eight bits. The data format of `byte` data is guaranteed; it is not changed when data is transmitted with RPC.
- The format of an integer type, character type, floating-point type, or the configuration type in which these types are combined may be converted if data is transmitted between hosts that use different data formats. If the data format should not be converted, transmit data as a `byte` type array.
- The efficiency of the `byte` type is higher than other data types since it is without format conversion.

void type (basic data type)

Format

```
void
```

Data type explanation

The following explains how to use the `void` type:

- Specify an operation type that returns no value, or indicate a parameter-free operation.

Error status type (basic data type)

Format

```
error_status_t
```

Data type specification

The error status type is predefined to maintain RPC communication status information.

Multi-language type (basic data type)

Format

```
ISO_LATIN_1
ISO_MULTI_LINGUAL
ISO_UCS
```

Data type explanation

With the multi-language type, the expressions of characters and character strings used in system files are predefined in conformance with the current and forthcoming international standards.

- The `char` type data may be converted to ASCII-EBCDIC if transmitted through RPC mechanism. The data format of a predefined multi-language type is not converted, because it consists of only `byte` type data (basic data type). Each data type is predefined as shown below.

```
typedef  byte  ISO_LATIN_1
typedef  struct  {
  byte  row,column;
}  ISO_MULTI_LINGUAL
typedef  struct  {
  byte  group,plane,row,column;
}  ISO_UCS
```

- With IDL-only TxRPC, the `char` type data is not converted to ASCII-EBCDIC. The definition of this type, therefore, has no meaning.

Structure (configuration data type)

Format 1

```
struct [tag]
{
 [ [struct_member_attribute, ...] ]  type_specifier declarator, ...;
}
```

Format 2

```
struct tag
```

Data type explanation

If `tag` is specified as a specifier in format 1, the sequence of member declaration procedures is expressed in an abbreviated format. This `tag` can be used as a specifier in subsequent format 2.

- `struct_member_attribute`

  There is no attribute that can be specified with the corresponding version.

6.8 Type declarators

This subsection explains type declarators used for IDL file declaration. The following type declarators can be used with OpenTP1 TxRPC:

- Array
- Character string
- Pointer

OpenTP1 IDL-only TxRPC restriction

Only one asterisk (*) can be used for the pointer.

The explanation format is as follows:

Format

Indicates the data type format.

Explanations of the type declarators

Explains about type declarators.

Array

Format

An IDL array is declared through the syntax of the `array_declarator` structure given below.

```
array_identifier  array_bounds_declarator...
```

Explanations of the type declarators

The following array type can be used:

- Fixed

  The array size is defined in the IDL. All array data items are transferred during a function call.

■ `array_bounds_declarator`

Specify each array dimension. The `array_bounds_declarator` for one-dimensional array must be in either of the following formats:

[lower..upper]: Specify a lower limit for `lower` and an upper limit for `upper`.

[size]: Specify `0` for a lower limit and `size-1` for an upper limit.

In the IDL, the normal value for `lower` is `0` only.

- An integer must be specified for the array limit. The array attribute can reference structure members and integer item parameters only.

Character string

Format

```
char
byte
```

Explanations of the type declarators

In the IDL, a character string is assumed as a one-dimensional array with the `string` attribute assigned. The array element type must be the following values:

- Member of the `byte` type
- Structure having all members predefined to be the `byte` type
- Type predefined to be the `char` or `byte` type

Pointer

Format

The following syntax is used for IDL pointer declaration.

```
* [*...] pointer_identifier
```

Explanations of the type declarators

Multiple asterisks set in the pointer operator indicate that there is a multiple-level indirect reference.

6.9 Attribute configuration language

The attribute configuration language cannot be used with IDL-only TxRPC.

6.10 IDL compiler (txidl command)

This section explains the syntax of the IDL compiler (`txidl` command) in the following format:

Format

Indicates the IDL compiler specification format.

Description

Indicates the IDL compiler facilities.

Arguments to be specified for argument

Indicates the arguments to be specified for argument.

Explanation

Explains about the IDL compiler.

Messages

Indicates messages output from the IDL compiler.

Related files

Indicates files related to the IDL compiler.

Note

Indicates the notes on the IDL compiler.

txidl (IDL compiler)

Format

```
txidl  filename  [argument]  ...
```

Description

The `txidl` command activates the TxRPC interface definition language compiler.

Arguments to be specified for argument

- `-cptype process_type`

  Specify a client process type. Specify either of the following values for the `process_type`:

  - `ndce`

This process uses the TP1/Server Base library.

- `nbet`

  This process uses the DCE library only.

If no value is specified, `ndce` is assumed. The program does not run if compiled with an incorrect process type specified. (For example, the program does not run if the TP1/Server Base library is incorporated in a stub compiled with `nbet` specified.)

- `-sptype process_type`

  Specify a server process type. The `process_type` is the same as the `-cptype`.

  If no value is specified, `ndce` is assumed. The program does not run if it is compiled with an incorrect process type specified. (For example, the program does not run if the TP1/Server Base library is incorporated in a stub compiled with `nbet` specified.)

- `-client file_type`

  Specify which client file is to be created. If this argument or `file_type` is not specified, the compiler creates all client files. Specify one of the following values for the `file_type`:

  - `none`

    No file is created.

  - `stub`

    Only stub files are created.

  - `all`

    Stub and client-created files are created.

- `-server file_type`

  Specify which server file is to be created. If this argument or `file_type` is not specified, the compiler creates all client files. The `file_type` is the same as the `-client`.

- `-cstub filename`

  Specify the pathname of the client stub.

  Do not specify an extension for the file name. The `txidl` compiler suffixes `.c` to a source file in C language. It suffixes `_cstub.c` to the file if the `-cstub` option is not used.

  When the client process type is `gateway` and the server process type is `dce`, two types of stub files are created. In this case, `B` is prefixed to the `filename` of the OpenTP1 stub file name.

- `-sstub filename`

  Specify the pathname of the server stub. Do not specify an extension for the file name. The `txidl` compiler suffixes `.c` to a source file in C language. It suffixes `_sstub.c` to the file if the `-sstub` option is not used.

- `-header header_file`

  Specify the pathname of the header file to be created.

  Do not specify an extension for the file name. For default, the `txidl` compiler suffixes `.h` to the base name of the IDL file.

- `-cconf conffile`

  Specify the pathname of the user service definition file or environment establishment file of the client program. If the `-cconf` option is not used, a file having the name with `C` prefixed to the base name of the IDL file is created. This option is valid only when the process type combination is IDL-only TxRPC. If this option is specified with any other process type combination, this option is simply ignored without causing an error.

- `-sconf conffile`

  Specify the pathname of the user service definition file of the server program. If the `-sconf` option is not used, a file having the name with `S` prefixed to the base name of the IDL file is created. This option is valid only when the process type combination is IDL-only TxRPC. If this option is specified with any other process type combination, this option is simply ignored without causing an error.

- `-out directory`

  Creates an output file under a specified directory. For default, the compiler creates an output file under the current directory.

  A path name specified in another option has priority regardless of the specification sequence.

- `-Idirectory`

  Specify the name of a directory containing the interface definition file to be imported. Multiple directories can be specified by specifying the additional `-Idirectory` argument on the command line. The compiler searches the directories in the sequence set in this argument.

  If one file is under multiple directories, the compiler imports the file that first appears.

  If this argument is omitted, the directories are searched in the following sequence:

  1. Current directory
  2. All specified directories
  3. System IDL directory (`$DCDIR/include`)

- `-no_def_idir`

  Specify this argument when the compiler is to search only the current directory for the import file. If this option is specified together with `-Idirectory`, the compiler searches only the directory specified by the user, but not the current and system directories.

- `-noconf`

  Specify this argument when the templates of OpenTP1 user service definition and environment establishment files are not to be created. This argument is valid only when the process type combination is only IDL-only TxRPC.

- `-noserver`

  Specify this argument when the template of the server program is not to be created. This argument is valid only when the process type combination is only IDL-only TxRPC.

- `-syntax_only`

  Specify this argument when only the syntax of the IDL file is to be checked but the file is not to be output.

Explanation

- The `txidl` command analyzes the interface definition written in the IDL and creates requisite files (including a header file, server stub file, client stub file, auxiliary file, and OpenTP1 definition file template).
- The IDL compiler searches each directory for the related ACF. For example, when a file named `source.idl` is compiled, the compiler automatically searches for a file named `source.acf`. It also searches for the imported IDL file (and related ACFs).

  The compiler searches for these files in the following sequence:

  1. Current directory

     The compiler always searches this directory unless the `-no_def_idir` and `-Idirectory` arguments are specified at the same time.

  2. Imported directory

     The compiler searches each directory specified for the `-Idirectory` argument.

  3. System IDL directory

     The compiler automatically imports `dctrpb.idl` in the system IDL directory. The compiler always searches this directory unless the `-no_def_idir` argument is specified.

4. Directory specified for the source file name

   If a directory is explicitly specified for the source IDL pathname, the corresponding ACF is searched under that directory.

- The `txidl` command automatically creates the OpenTP1 definition file with IDL-only TxRPC. It can be designed not to create the file by specifying an option in the `txidl` command.
- If the operation name is changed, the OpenTP1 definition file also needs to be re-created.

Messages

The `txidl` compiler outputs the three types of messages listed below. For details on the messages, see the indicated manuals.

1. Messages output by the `txidl` compiler

   See the manual *OpenTP1 Messages*.

2. Messages output by DCE idl activated by the `txidl` compiler

   See the corresponding DCE manuals.

3. Messages output by `cpp` or `cc` activated by DCE idl.

   See the manual corresponding to each command.

Related files

Files related to IDL-only TxRPC are as follows:

`$DCDIR/bin/txidl`: IDL compiler

`$DCDIR/include/dctrpb.idl`: System IDL file

`$DCDIR/include/dctrp.h`: Header file

Notes

- The IDL compiler creates ANSI C code. No warning message is returned while a stub is being compiled by the C compiler. However, the following messages may be posted if the C compiler does not completely conform to the ANSI C specification:

```
warning: & before array or function: ignored
warning: enumeration type clash, operator=
```

- Place a space between the option and parameter.

  Example:

```
-out xxx (Do not write as -outxxx.)
```

- The file names listed below are reserved by the IDL compiler. If the IDL file is named using any of these file names, the operation is undefined.

  `iovector.idl`, `lbase.idl`, `nbase.idl`, `ncastat.idl`, `rpc.idl`, `rpcbase.idl`, `rpcpvt.idl`, `rpcsts.idl`, `rpctypes.idl`, `twr.idl`, `uuid.idl`, or `dctrpb.idl`

- This version does not support RPC TxRPC. Therefore, even if you specify `nbet` as the process type in the `-cptype` option and `-sptype` option of the `txidl` command, the generated stub file cannot be used.

6.11 TxRPC error codes

This section explains errors returned from the OpenTP1 TxRPC system service.

The table below lists TxRPC error codes. The table also describes the equivalent return values returned from the function `dc_rpc_call()`. When you create error handling processing for IDL-only TxRPC, see the description of these equivalent return values for reference.

Table 6-1: TxRPC error codes

Error code	Meaning
`txrpc_x_not_in_transaction`	The operation with `transaction_mandatory` specified was called from outside a global transaction.
`txrpc_x_no_tx_open_done`	When the manager was called with an OpenTP1 TxRPC system service, the operation was executed although the function `tx_open()` was not called.
`DCTRPER_PROTO`	A protocol error occurred.
`rpc_s_comm_failure`	A communication-related error occurred. Equivalent to the following return values of the function `dc_rpc_call()`: • `DCRPCER_SYSERR` • `DCRPCER_SYSERR_RB` • `DCRPCER_SYSERR_AT_SERVER` • `DCRPCER_SYSERR_AT_SERVER_RB` • `DCRPCER_SERVICE_TERMINATING` • `DCRPCER_SERVICE_NOT_UP` • `DCRPCER_SERVICE_CLOSED` • `DCRPCER_OLTF_NOT_UP` • `DCRPCER_OLTF_INITIALIZING`
`rpc_s_no_memory`	Memory became insufficient. Equivalent to the following return value of the function `dc_rpc_call()`: • `DCRPCER_NO_BUFS`
`rpc_s_fault_remote_no_memory`	Server memory became insufficient. Equivalent to the following return values of the function `dc_rpc_call()`: • `DCRPCER_NO_BUFS_RB` • `DCRPCER_NO_BUFS_AT_SERVER`
`rpc_s_call_timeout`	A timeout occurred. Equivalent to the following return value of the function `dc_rpc_call()`: • `DCRPCER_TIMED_OUT`
`rpc_s_in_args_too_big`	A value specified for an argument is too big. Equivalent to the following return value of the function `dc_rpc_call()`: • `DCRPCER_MESSAGE_TOO_BIG`

Error code	Meaning
rpc_s_entry_not_found	No service entry found. Equivalent to the following return values of the function dc_rpc_call(): • DCRPCER_NO_SUCH_SERVICE_GROUP • DCRPCER_NO_SUCH_SERVICE
rpc_s_mgmt_op_disallowed	The server is a socket reception server, and it cannot receive the service request. Or the server is protected with the OpenTP1 security facility, and the client has no access to the server. Equivalent to the following return value of the function dc_rpc_call(): • DCRPCER_SERVER_BUSY
rpc_s_binding_has_no_auth	The server is using the OpenTP1 security facility. An access error occurred with the security facility. Equivalent to the following return value of the function dc_rpc_call(): • DCRPCER_SECCHK
rpc_s_fault_unspec	In an OpenTP1 system, an error equivalent to one of the following return values of the function dc_rpc_call() occurred. • DCRPCER_TESTMODE • DCRPCER_INVALID_REPLY • DCRPCER_REPLY_TOO_BIG • DCRPCER_REPLY_TOO_BIG_RB Alternatively, marshaling/unmarshaling failed, or communications data was destroyed.
rpc_s_unknown_stub_rtl_if_vers	Version in the OpenTP1 library is different.
rpc_s_unknown_if	Version in the interface definition is different.

Chapter
7. Coding Samples

This chapter gives coding samples for application programs (UAPs).

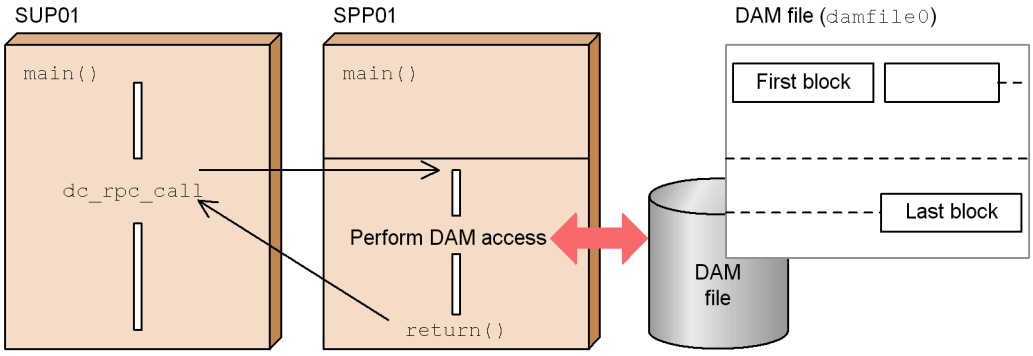
This chapter presents coding samples for application programs (7.1 to 7.4) in K&R (Classic C) C language.

This chapter contains the following sections:

7.1 Coding samples for client/server configuration UAPs (SUP, SPP DAM access)

The figure below shows an example of a client/server configuration UAP.

Figure 7-1: Client/Server configuration UAP sample (DAM access)



Explanation

DAM file `damfile0` contains a control section in its first block and data records in the second and subsequent blocks. During service processing, the first block is read (the function `dc_dam_read()`) and is updated (the function `dc_dam_rewrite()`), then the second and subsequent blocks are directly updated using the function `dc_dam_write()`.

This section presents a coding example based on the configuration sample shown in the figure.

(1) SUP sample

The following shows a coding example for an SUP.

```
 10  /*
 20   * SUP01
 30   */
 40  #include <stdio.h>
 50  #include <string.h>
 60  #include <dcrpc.h>
 70  #include <dctrn.h>
 80
 90  main()
100  {
110  /*
120   * Define variables
```

```
130   */
140    static char   in_buf[1024];
150    static DCLONG in_buf_len;
160    static char   out_buf[1024];
170    static DCLONG out_buf_len;
180    int rc;
190  /*
200   * RPC-OPEN (start the UAP)
210   */
220    rc = dc_rpc_open(DCNOFLAGS);
230    /* Prepare to use various OpenTP1 functions  */
235    /* (initialize each function)                */
240    if(rc != DC_OK) {
250      printf("SUP01:dc_rpc_open failed. CODE = %d \n",rc);
260      goto PROG_END;
270    }
280  /*
290   * ADM-COMPLETE (report completion
295   * of user server start processing)
300   */
310    rc = dc_adm_complete(DCNOFLAGS);
320    if(rc != DC_OK){
330      printf("SUP01:dc_adm_complete failed."
335             "CODE = %d \n",rc);
340      goto PROG_END;
350    }
360  /*
370   * TRN_BEGIN (start the transaction)
380   */
390    rc = dc_trn_begin();
400    if(rc != DC_OK) {
410      printf("SUP01:dc_trn_begin failed. CODE = %d \n",rc);
420      goto TRAN_END;
430    }
440    /*
450     * RPC-CALL (request a remote service)
460     */
470    strcpy(in_buf,"SUP01:DATA OpenTP1!!");
480    in_buf_len = strlen(in_buf) + 1;
490    out_buf_len = 1024;
500    rc = dc_rpc_call("spp01grp","svr01",
505                     in_buf,&in_buf_len,
510                     out_buf,&out_buf_len,DCNOFLAGS);
520    if(rc != DC_OK) {
530      printf("SUP01:Service request failed. "
535             "CODE = %d \n",rc);
540      goto TRAN_END;
550    }
```

```
560    printf("SUP01:SERVICE FUNCTION RETURN = %s\n",
565            out_buf);
570  /*
580   * TRN-UNCHAINED-COMMIT (commit in unchained mode)
590   */
600    TRAN_END:
610    rc = dc_trn_unchained_commit();
620    if(rc != DC_OK) {
630      printf("SUP01:dc_trn_unchained_commit failed. "
635             "CODE = %d \n",rc);
640    }
650  /*
660   * RPC-CLOSE (terminate the UAP)
670   */
680    PROG_END:
690    dc_rpc_close(DCNOFLAGS);
700    printf("SUP01:Processing is finished.\n");
710    exit(0);
720  }
```

(2) SPP sample (main function)

The following shows a coding example for the SPP main function.

```
 10  /*
 20   * SPP01 main function
 30   */
 40  #include <stdio.h>
 50  #include <dcrpc.h>
 60  #include <dcdam.h>
 70  #define DAMFILE "damfile0"
 80
 90  int damfd;   /* damfile file-id */
100
110  main()
120  {
130  /*
140   * Define area for storing return value
150   */
160    int rc;
170  /*
180   * RPC-OPEN (start the UAP)
190   */
200    rc = dc_rpc_open(DCNOFLAGS);
210    if(rc != DC_OK) {
220      printf("SPP01:dc_rpc_open failed. CODE = %d \n",rc);
230      goto PROG_END;
240    }
250  /*
260   * DAM-OPEN (open a logical file)
```

```
270   */
280    rc = dc_dam_open(DAMFILE,DCDAM_BLOCK_EXCLUSIVE);
290    if(rc < DC_OK) {
300      printf("SVR01:dc_dam_open failed. CODE = %d \n",rc);
310      goto DAM_END;
320    }
330    damfd = rc;
340  /*
350   *  RPC-MAINLOOP (start the SPP service)
360   */
370    printf("SPP01:mainloop begins.\n");
380    rc = dc_rpc_mainloop(DCNOFLAGS);
390    if(rc != DC_OK) {
400      printf("SPP01:dc_rpc_mainloop \
410          failed. CODE = %d \n",rc);
420    }
430  /*
440   * DAM-CLOSE (close the logical file)
450   */
460    DAM_END:
470    rc = dc_dam_close(damfd,DCNOFLAGS);
480    if(rc != DC_OK) {
490      printf("SVR01:dc_dam_close failed. CODE = %d\n",rc);
500    }
510  /*
520   * RPC-CLOSE (terminate the UAP)
530   */
540    PROG_END:
550    dc_rpc_close(DCNOFLAGS);
560    printf("SPP01:The SPP service processing is "
565           "terminated. \n");
570    exit(0);
580  }
```

(3) SPP sample (service function)

The following shows a coding example for the SPP service function.

```
 10  /*
 20   * SVR01 service function
 30   */
 40  #include <stdio.h>
 50  #include <string.h>
 60  #include <dcrpc.h>
 70  #include <dcdam.h>
 80  #define DAMFILE "damfile0"
 90  #define DAM_BLK_SIZE 504
100  #define REWRITE_LEN 19
110  extern int damfd;
120
```

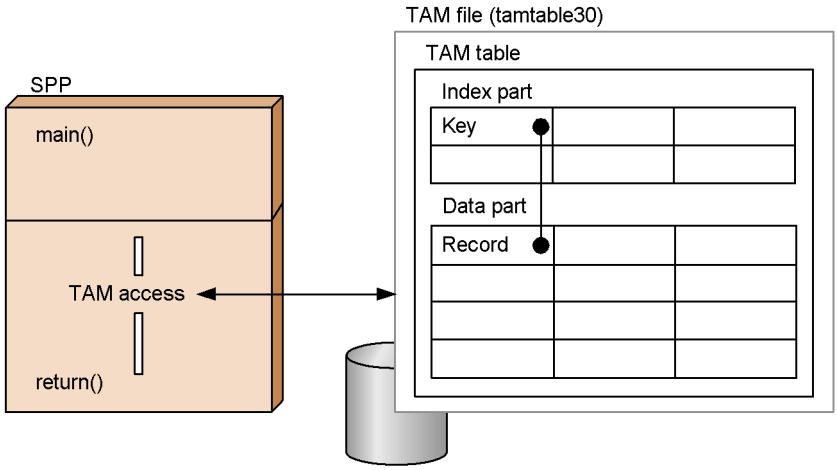
```
130  void svr01(in_data,in_leng,out_data,out_leng)
140    char   *in_data;
150    DCLONG *in_leng;
160    char   *out_data;
170    DCLONG *out_leng;
180  {
190  /*
200   * Define variables
210   */
220    static struct DC_DAMKEY keyptr;
230    static char *damc_buf;
240    static char dam_cntl_buf[DAM_BLK_SIZE];
250    static char write_buf[DAM_BLK_SIZE];
260    struct dam_cntl_p {
270      int w_point;
280      char rewrite_data[REWRITE_LEN];
290    } *dam_cntl_p;
300    int rc;
310    int write_size;
320    int rewrite_size;
330    int damc_buf_size;
340
350    keyptr.fstblkno = 0;
360    keyptr.endblkno = 0;
370    damc_buf_size = DAM_BLK_SIZE;
380    printf("SVR01:Start of processing\n");
390  /*
400   * DAM_READ(read logical file blocks)
410   */
420    rc = dc_dam_read(damfd,&keyptr,1,dam_cntl_buf,
430    damc_buf_size,DCDAM_MODIFY);
440    if(rc != DC_OK) {
450      printf("SVR01:dc_dam_read failed. CODE = %d \n",rc);
460      strcpy(out_data,"SVR01:DAM READ FAILED");
470      *out_leng = strlen(out_data);
480      goto PROG_END;
490    }
500  /*
510   * DAM_WRITE (write to logical file blocks)
520   * DAM_REWRITE (update logical file blocks)
530   */
540    DAM_WRITE:
550    dam_cntl_p = (struct dam_cntl_p *)dam_cntl_buf;
560    write_size = DAM_BLK_SIZE;
570    memcpy(write_buf,in_data,*in_leng);
580    dam_cntl_p->w_point = dam_cntl_p->w_point + 1;
590    keyptr.fstblkno = dam_cntl_p->w_point;
600    keyptr.endblkno = 0;
```

```
610    rc = dc_dam_write(damfd,&keyptr,1,write_buf,
620    write_size,DCNOFLAGS);
630    if(rc != DC_OK) {
640      if(rc == DCDAMER_BNOER) {
650        dam_cntl_p->w_point = 0;
660        goto DAM_WRITE;
670      }
680      printf("SVR01:dc_dam_write failed. "
685             "CODE = %d \n",rc);
690      strcpy(out_data,"SVR01;DAM WRITE FAILED");
700      *out_leng = strlen(out_data);
710      goto PROG_END;
720    }
730    keyptr.fstblkno = 0;
740    keyptr.endblkno = 0;
750    damc_buf_size = DAM_BLK_SIZE;
760    sprintf(dam_cntl_p->rewrite_data,
765            "REWRITE COMPLETE\n");
770    rc = dc_dam_rewrite(damfd,&keyptr,1,dam_cntl_buf,
780    damc_buf_size,DCDAM_UPDATE);
790    if(rc != DC_OK) {
800      printf("SVR01:dc_dam_rewrite failed. "
805             "CODE = %d\n",rc);
810      strcpy(out_data,"SVR01:DAM REWRITE FAILED");
820      *out_leng = strlen(out_data);
830    }
840    strcpy(out_data,"SVR01:PROCESS COMPLETE");
850    *out_leng = strlen(out_data);
860    PROG_END:
870    printf("SVR01:Processing is terminated.\n");
880    return;
890  }
```

7.2 Coding samples for client/server configuration UAPs (SPP TAM access)

The figure below shows an example of a client/server configuration UAP. This section presents only an SPP coding sample. This example assumes that the same SUP as in *7.1 Coding samples for client/server configuration UAPs (SUP, SPP DAM access)* requests this SPP for service.

Figure 7-2: Client/server configuration UAP sample (TAM access)



This section presents a coding example based on the configuration sample shown in the figure.

(1) SPP sample (main function)

The following shows a coding example for the SPP main function.

```
 10  /*
 20   * spp01 main function
 30   */
 40  #include <stdio.h>
 50  #include <dcrpc.h>
 60  #include <dctam.h>
 70  #define  TAMTABLE    "tamtable30"
 80
 90  long  tamfd ;      /* tamfile file-id  */
100
110  main()
120  {
130
140  /*
```

```
150   * Define a return code storage variable
160   */
170    int    rcd ;
180  /*
190   * RPC-OPEN (start the UAP)
200   */
210    rcd = dc_rpc_open(DCNOFLAGS) ;
220    if(rcd != DC_OK) {
230      printf("SPP01:dc_rpc_open failed. "
235             "code = %d \n", rcd) ;
240      goto PROG_END ;
250    }
260  /*
270   * TAM-OPEN (open a TAM table)
280   */
290    rcd = dc_tam_open(TAMTABLE, DCTAM_REC_EXCLUSIVE) ;
300    if(rcd <= 0) {
310      printf("SVR01:dc_tam_open failed. "
315             "code = %d \n", rcd) ;
320      goto TAM_END ;
330    }
340    tamfd = (long)rcd ;
350  /*
360   * RPC-MAINLOOP (start the SPP service)
370   */
380    rcd = dc_rpc_mainloop(DCNOFLAGS) ;
390    if(rcd != DC_OK) {
400      printf("SPP01:dc_rpc_mainloop failed. "
405             "code = %d \n", rcd) ;
410    }
420  /*
430   * TAM-CLOSE (close the TAM table)
440   */
450    rcd = dc_tam_close(tamfd, DCNOFLAGS) ;
460    if(rcd != DC_OK) {
470      printf("SVR01:dc_tam_close failed. "
475             "code = %d \n", rcd) ;
480    }
490  TAM_END :
500  /*
510   * RPC-CLOSE (terminate the UAP)
520   */
530    dc_rpc_close(DCNOFLAGS) ;
540  PROG_END :
550    printf("SPP01:The SPP service processing is "
555           "terminated. \n") ;
560    exit(0) ;
570  }
```

(2) SPP sample (service function)

The following shows a coding example for the SPP service function.

```
 10  /*
 20   * srv01 service function
 30   */
 40  #include <stdio.h>
 50  #include <string.h>
 60  #include <dctam.h>
 70  #define  TAM_REC_SIZE  128
 80
 90  extern  long tamfd ;   /* tamfile file-id */
100
110  void svr01(in_data, in_leng, out_data, out_leng)
120    char *in_data ;
130    long *in_leng ;
140    char *out_data ;
150    long *out_leng ;
160  {
170
180  /*
190   * Define variables
200   */
210    static struct DC_TAMKEY keyptr ;
220    static char   *tamc_buf ;
230    static char   tam_cntl_buf[TAM_REC_SIZE] ;
240    static char   write_buf[TAM_REC_SIZE] ;
250    struct tam_cntl_p {
260                       char keyname[10] ;
270                       char filler[118] ;
280                      } *tam_cntl_p ;
290    int    rcd ;
300    int    write_size ;
310    int    tamc_buf_size ;
320    static char keypar[4][10] = {
330                       { 0x00, 0x00, 0x00, 0x00, 0x00,
340                         0x00, 0x00, 0x00, 0x00, 0x01} ,
350                       { 0x00, 0x00, 0x00, 0x00, 0x00,
360                         0x00, 0x00, 0x00, 0x00, 0x02} ,
370                       { 0x00, 0x00, 0x00, 0x00, 0x00,
380                         0x00, 0x00, 0x00, 0x00, 0x03} ,
390                       { 0x00, 0x00, 0x00, 0x00, 0x00,
400                         0x00, 0x00, 0x00, 0x00, 0x04} ,
410                                  } ;
420    printf("SVR01:Start of processing \n") ;
430  /*
440   * TAM_READ (read the first record from the TAM table)
450   */
```

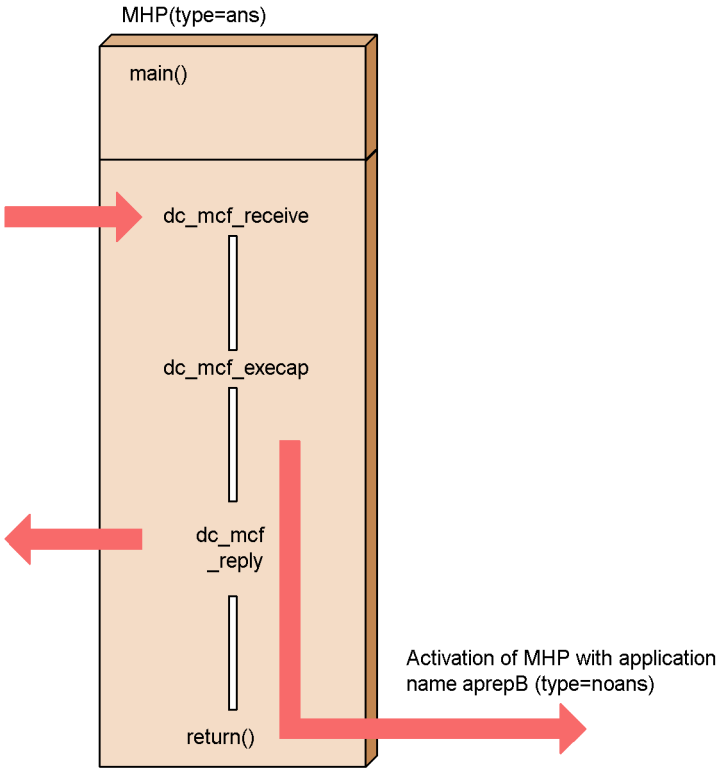
```
460    keyptr.keyname = keypar[0] ;
470    tamc_buf_size = TAM_REC_SIZE ;
480    rcd = dc_tam_read(tamfd, &keyptr, 1, tam_cntl_buf,
490                      tamc_buf_size,
495                      DCTAM_EQLSRC | DCTAM_MODIFY) ;
500    if(rcd != DC_OK) {
510      printf("SVR01:dc_tam_read failed. "
515             "code = %d \n", rcd) ;
520      strcpy(out_data, "SVR01:TAM READ FAILED") ;
530      *out_leng = strlen(out_data) ;
540      goto PROG_END ;
550    }
560  /*
570   * TAM_REWRITE (update the first record of TAM table
575   *              on the assumption of search)
580   */
590    tam_cntl_p = (struct tam_cntl_p *)tam_cntl_buf ;
600    memcpy(tam_cntl_p->filler, in_data, *in_leng) ;
610    rcd = dc_tam_rewrite(tamfd, &keyptr, 1, tam_cntl_buf,
620                         tamc_buf_size, DCNOFLAGS) ;
630    if(rcd != DC_OK) {
640      printf("SVR01:dc_tam_rewrite failed. "
645             "code = %d \n", rcd) ;
650      strcpy(out_data, "SVR01:TAM REWRITE FAILED") ;
660      *out_leng = strlen(out_data) ;
670      goto PROG_END ;
680    }
690  /*
700   * TAM_WRITE (update the second record of TAM table)
710   */
720    keyptr.keyname = keypar[1] ;
730    tam_cntl_p = (struct tam_cntl_p *)write_buf ;
740    memcpy(tam_cntl_p->keyname, keypar[1], 10) ;
750    memcpy(tam_cntl_p->filler, in_data, *in_leng) ;
760    write_size = TAM_REC_SIZE ;
770    rcd = dc_tam_write(tamfd, &keyptr, 1, tam_cntl_p,
780                               write_size, DCTAM_WRITE) ;
790    if(rcd != DC_OK) {
800      printf("SVR01:dc_tam_write failed. "
805             "code = %d \n", rcd) ;
810      strcpy(out_data, "SVR01:TAM WRITE FAILED") ;
820      *out_leng = strlen(out_data) ;
830      goto PROG_END ;
840    }
850  /*
860   * TAM_READ (read the third record from the TAM table)
870   */
880    keyptr.keyname = keypar[2] ;
```

```
 890    tamc_buf_size = TAM_REC_SIZE ;
 900    rcd = dc_tam_read(tamfd, &keyptr, 1, tam_cntl_buf,
 910                      tamc_buf_size,
 905                      DCTAM_EQLSRC | DCTAM_MODIFY) ;
 920    if(rcd != DC_OK) {
 930      printf("SVR01:dc_tam_read failed. "
 935             "code = %d \n", rcd) ;
 940      strcpy(out_data, "SVR01:TAM READ FAILED") ;
 950      *out_leng = strlen(out_data) ;
 960      goto PROG_END ;
 970    }
 980  /*
 990   * TAM_READ_CANCEL (cancel the search for the third
 995   * record of the TAM table)
1000   */
1010    rcd = dc_tam_read_cancel(tamfd, &keyptr,
1015                             1, DCNOFLAGS) ;
1020    if(rcd != DC_OK) {
1030      printf("SVR01:dc_tam_read_cancel failed. "
1040             "code = %d \n",rcd) ;
1050      strcpy(out_data, "SVR01:TAM READ CANCEL FAILED") ;
1060      *out_leng = strlen(out_data) ;
1070      goto PROG_END ;
1080    }
1090  /*
1100   * TAM_delete (delete the fourth record of the
1105   *             TAM table)
1110   */
1120    keyptr.keyname = keypar[3] ;
1130    rcd = dc_tam_delete(tamfd, &keyptr, 1,
1140                               NULL, 0, DCTAM_NOOUTREC) ;
1150    if(rcd != DC_OK) {
1160      printf("SVR01:dc_tam_delete failed. "
1165             "code = %d \n", rcd) ;
1170      strcpy(out_data, "SVR01:TAM DELETE FAILED") ;
1180      *out_leng = strlen(out_data) ;
1190      goto PROG_END ;
1200    }
1210    strcpy(out_data, "SVR01:PROCESS COMPLETE") ;
1220    *out_leng = strlen(out_data) ;
1230  PROG_END :
1240    printf("SVR01:Processing is terminated.\n") ;
1250    return ;
1260  }
```

7.3 Coding samples for message exchange configuration UAPs (MHP)

The figure below shows an example of a message exchange UAP.

Figure 7-3: Message exchange configuration UAP sample (MHP)



This section presents a coding example based on the configuration sample shown in the figure.

(1) MHP sample (main function)

The following shows a coding example for the MHP main function.

```
10  /*
20   * MHP main function
30   */
40  #include <stdio.h>
50  #include <dcrpc.h>
60  #include <dcmcf.h>
70
```

```
 80  main()
 90  {
100     int rtn_cod ;
110
120     printf("******  RPC OPEN   *****\n") ;
130  /*
140   * RPC-OPEN (start the UAP)
150   */
160     rtn_cod = dc_rpc_open(DCNOFLAGS) ;
170     if(rtn_cod != DC_OK) {
180       printf("dc_rpc_open failed !! CODE = %d \n",
185              rtn_cod) ;
190       goto PROG_END ;
200     }
210
220     printf("******  MCF OPEN   *****\n") ;
230  /*
240   * MCF-OPEN (open the MCF environment)
250   */
260     rtn_cod = dc_mcf_open(DCNOFLAGS) ;
270     if(rtn_cod != DC_OK) {
280       printf("dc_mcf_open failed !! CODE = %d \n",
285              rtn_cod) ;
290       goto PROG_END ;
300     }
310
320     printf("******  MCF MAINLOOP   *****\n") ;
330  /*
340   * MCF-MAINLOOP (start the MHP service)
350   */
360     rtn_cod = dc_mcf_mainloop(DCNOFLAGS) ;
370     if(rtn_cod != DC_OK) {
380       printf("dc_mcf_mainloop failed !! CODE = %d \n",
385              rtn_cod) ;
390     }
400
410     printf("******  MCF CLOSE   *****\n") ;
420  /*
430   * MCF-CLOSE (close the MCF environment)
440   */
450     dc_mcf_close(DCNOFLAGS) ;
460
470  PROG_END :
480     printf("******  RPC CLOSE   *****\n") ;
490  /*
500   * RPC-CLOSE (terminate the UAP)
510   */
520     dc_rpc_close(DCNOFLAGS) ;
```

```
530     exit(0) ;
540  }
```

(2) MHP sample (service function)

The following shows a coding example for the MHP service function.

```
 10  /*
 20   * MHP service function
 30   */
 40  #include <stdio.h>
 50  #include <sys/types.h>
 60  #include <dcmcf.h>
 70  #include <dcrpc.h>
 80
 90  void svrA()
100  {
110       DCLONG action ;
120       DCLONG commform ;
130       DCLONG opcd ;
140       DCLONG active ;
150       char recvdata[1024] ;
160       DCLONG rdataleng ;
170       DCLONG time ;
180       DCLONG inbufleng ;
190       int  rtn_cod ;
200       DCLONG cdataleng ;
210       char termnam[10] ;
220       static char execdata[32] = "  SVRA EXECAP DATA" ;
230       static char senddata[32] = "  SVRA REPLY DATA1" ;
240       static char resv01[9] = "\0" ;
250       static char resv02[9] = "\0" ;
260       static char resv03[9] = "\0" ;
270       static char apnam[9] = "aprepB" ;
280
290       printf("*****   UAP START   *****\n") ;
300
310       printf("*****   MCF RECEIVE   *****\n") ;
320  /*
330   * MCF-RECEIVE (receive messages)
340   */
350       action = DCMCFFRST ;
360       commform = DCNOFLAGS ;
370       inbufleng = sizeof(recvdata) ;
380       rtn_cod = dc_mcf_receive(action, commform,
385                         termnam, resv01, recvdata,
390                         &rdataleng, inbufleng, &time) ;
400       if(rtn_cod != DCMCFRTN_00000) {
410  /*
420   * MCF-ROLLBACK (error processing)
```

```
430   */
440       printf("dc_mcf_receive failed !! CODE = %d \n",
445              rtn_cod) ;
450       rtn_cod = dc_mcf_rollback(DCMCFNRTN) ;
460       }
470
480       printf("*****   MCF EXECAP   *****\n") ;
490   /*
500    * MCF-EXECAP (start the application program)
510    */
520       action = DCMCFEMI|DCMCFJUST ;
530       commform = DCNOFLAGS ;
540       active = 0 ;
550       cdataleng = 16 ;
560       rtn_cod = dc_mcf_execap(action, commform, resv01,
570                 active, apnam, execdata, cdataleng) ;
580       if(rtn_cod != DCMCFRTN_00000) {
590   /*
600    * MCF-ROLLBACK (error processing)
610    */
620       printf("dc_mcf_execap failed !! CODE = %d \n",
625              rtn_cod) ;
630       rtn_cod = dc_mcf_rollback(DCMCFNRTN) ;
640       }
650
660       printf("*****   MCF REPLY   *****\n") ;
670   /*
680    * MCF-REPLY (send a response message)
690    */
700       action = DCMCFEMI ;
710       commform = DCNOFLAGS ;
720       opcd = DCNOFLAGS ;
730       cdataleng = 16 ;
740       rtn_cod = dc_mcf_reply(action, commform,
745                 resv01, resv02, senddata,
750                 cdataleng, resv03, opcd) ;
760       if(rtn_cod != DCMCFRTN_00000) {
770   /*
780    * MCF-ROLLBACK (error processing)
790    */
800       printf("dc_mcf_reply failed !! CODE = %d \n",
805              rtn_cod) ;
810       rtn_cod = dc_mcf_rollback(DCMCFNRTN) ;
820       }
830   }
```

7.4 Coding samples for X/Open-compliant UAPs

7.4.1 XATMI interface samples

(1) Request/response service paradigm sample

(a) Outline of processing

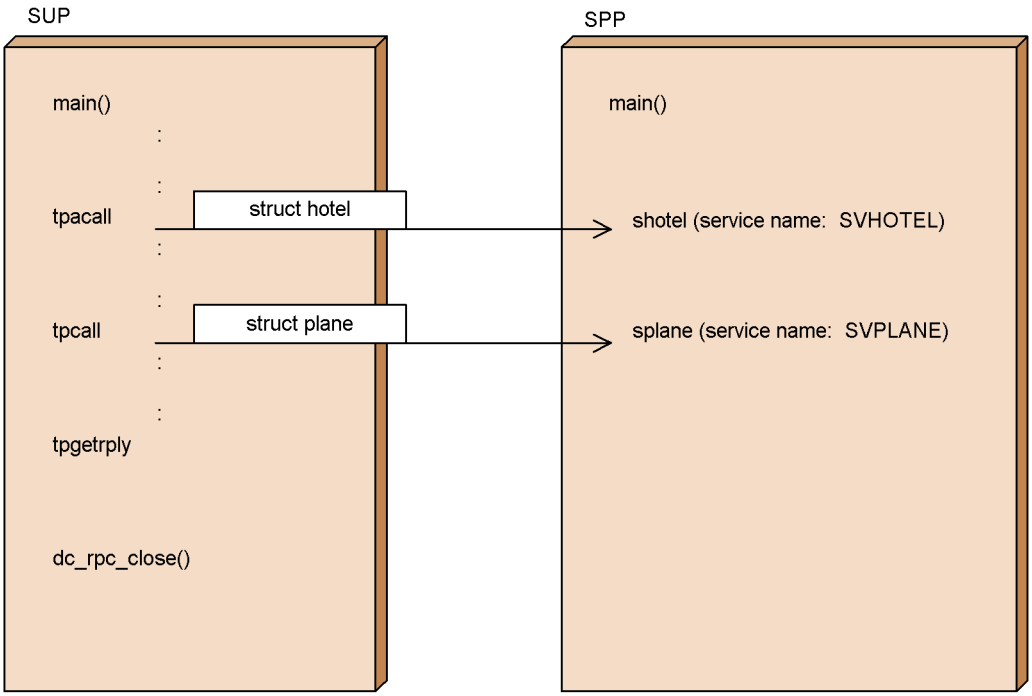
The processing of the sample here is outlined below.

A service for checking hotel room availability and a service for checking airplane seat availability are called from the SUP. The first service receives responses asynchronously, whereas the second service receives responses synchronously.

(b) UAP configuration

The figure below shows the configuration of the sample UAP.

Figure 7-4: Communication of request/response services receiving responses synchronously



(c) Typed buffers used

The following shows the structure of typed buffers used for communication.

```
struct hotel {
   long date;
    char plane[128];
    char hname[128];
   long status;
 }
```

```
struct plane {
   long date;
    char dest;
   long departure;
   long status;
 }
```

(d) SUP sample

- XATMI interface definition sample

  The following shows the XATMI interface definition of the SUP for the sample request/response service.

```
 10  /* Example of XATMI interface definition of SUP */
 15  /* (rrsup.def file)                              */
 20  called_servers = { "rrspp.def" };
```

- SUP coding sample

  The following shows a coding example for the SUP used in the example of request/response service.

```
 10  /* Example of SUP (rrsup.c file) */
 20  #include <stdio.h>
 30  #include <dcrpc.h>
 40  #include <xatmi.h>
 50  #include <dcadm.h>
 60  /*
 70   * XATMI stub header file
 80   */
 90  #include "rrsup_stbx.h"
100  main()
110  {
120  /*
130   * Define variables
140   */
150    struct hotel *hptr;
160    struct plane *pptr;
170    struct  errmsg *werrmsg ;
180    int  hlen, plen ;
190    int  cd ;
200    int rc;
210  /*
220   * RPC-OPEN (start the UAP)
230   */
240      rc = dc_rpc_open(DCNOFLAGS);
250      if(rc != DC_OK){
260          printf("dc_rpc_open failed.\
270                  ERROR CODE = %d \n", rc);
280          goto PROG_END;
```

```
290       }
300   /*
310    * ADM-COMPLETE (report completion of user
315    *               server start processing)
320    */
330       rc = dc_adm_complete(DCNOFLAGS);
340       if(rc != DC_OK){
350           printf("dc_adm_complete failed.\
360                   ERROR CODE = %d \n",  rc);
370           goto PROG_END;
380       }
390   /*
400    * TPALLOC (allocate typed buffer)
410    */
420     /* For hotel room availability search service */
430       hptr = (struct hotel *)tpalloc("X_COMMON",
435                                      "hotel", 0);
440       if(hptr == NULL){
450           printf("tpalloc failed.\
460                   ERROR CODE = %d \n", tperrno);
470           goto PROG_END;
480       }
490     /* For airplane seat availability */
500       pptr = (struct plane *)tpalloc("X_COMMON",
505                                      "plane", 0);
510       if(pptr == NULL){
520           printf("tpalloc failed.\
530                   ERROR CODE = %d \n", tperrno);
540           goto PROG_END;
550       }
560   /*
570    * Set data
580    */
590     hptr->date = 940415 ;
600     strcpy(hptr->place, "SAPPORO") ;
610     strcpy(hptr->hname, "PRINCE") ;
620     hptr->status = 0 ;
630     pptr->date = 940415 ;
640     strcpy(pptr->dest, "CHITOSE") ;
650     pptr->departure = 1540 ;
660     pptr->status = 0 ;
670   /*
680    * TPACALL (send a service request)
690    */
700       cd = tpacall("SVHOTEL", (char *) hptr, 0, 0);
710       if(cd == -1){
720          printf("The hotel room availability search "
725                 "service call failed.\
```

```
730                 ERROR CODE = %d \n", tperrno);
740            goto PROG_END;
750        }
760        printf("The hotel room availability search "
765               "service call was successful.\n");
770     /*
780      * TPCALL (send a service request and then wait for
785      *         a response)
790      */
800        rc = tpcall("SVPLANE", (char *) pptr, 0,
805                    (char **) &pptr, &plen, 0);
810        if(rc != 0){
820            if(tperrno == TPESVCFAIL){
830               werrmsg = (struct errmsg *) pptr ;
840               printf("%s ERROR CODE = %d USER CODE = %d\n",
850                     werrmsg->errmessage, tperrno, tpurcode);
860               goto PROG_END ;
870            }else{
880               printf("The airplane seat availability "
885                      "search service call failed. "
890                      "ERROR CODE = %d", tperrno);
900               goto PROG_END;
910            }
920        }
930        printf("A response to the airplane seat "
935               "availability search service call was "
937               "received successfully.\n");
940        if(pptr->status == 1){
950            printf("Airplane seat availability: Full \n");
960        } else {
970            printf("Airplane seat availability: "
975                   "Available \n");
980        }
990     /*
1000     * TPGETRPLY (receive a response)
1010     */
1020        rc = tpgetrply(&cd, (char **) &hptr, &hlen, 0);
1030        if(rc != 0){
1040            if(tperrno == TPESVCFAIL){
1050               werrmsg = (struct errmsg *) hptr ;
1060               printf("%s ERROR CODE = %d USER CODE = %d\n",
1070                     werrmsg->errmessage, tperrno, tpurcode);
1080               goto PROG_END ;
1090            }else{
1100               printf("The hotel room availability search "
1105                      "service failed. ERROR CODE = %d",
1110                       tperrno);
1120               goto PROG_END;
```

```
1130          }
1140      }
1150      printf("A response to the hotel room availability "
155          "search service was received successfully. \n");
1160      if(hptr->status == 1){
1170          printf("Hotel room availability: Full \n");
1180      } else {
1190          printf("Hotel room availability: Available \n");
1200      }
1210  /*
1220   * Release the typed buffer
1230   */
1240      tpfree((char *) hptr);
1250      tpfree((char *) pptr);
1260  /*
1270   * RPC-CLOSE (terminate the UAP)
1280   */
1290      PROG_END:
1300      dc_rpc_close(DCNOFLAGS);
1310      printf("Thank you for using our service.\n");
1320      exit(0);
1330  }
```

- User service definition sample

  The following shows a user service definition example for the SUP that was presented in the example of the request/response service.

```
10  # Example of the user service definition (rrsup file)
20  set  module              = "rrsup"
30  set  receive_from        = none
40  set  trn_expiration_time = 180
50  set  trn_expiration_time_suspend = Y
```

(e) SPP sample

- XATMI interface definition sample

  The following shows an XATMI interface definition example for the SPP that was presented in the example of the request/response service.

```
10  /* Example of XATMI interface definition */
15  /* (rrspp.def file) */
20  X_COMMON hotel {
30      long   date;
40      char   place[128];
50      char   hname[128];
60      long   status;
70  };
80  X_COMMON plane {
90      long   date;
```

```
100      char   dest[128];
110      long   departure;
120      long   status;
130  };
140  X_COMMON errmsg {
150      char   errmessage[128];
160  };
170  service shotel(X_COMMON hotel) ;
180  service splane(X_COMMON plane) ;
```

- SPP coding sample (main function)

  The following shows a coding example (main function) of the SPP that was presented in the example of the request/response service.

```
 10  /* Example of SPP main function (rrspp.c file) */
 20  #include <stdio.h>
 30  #include <dcrpc.h>
 40  #include <xatmi.h>
 50  #include <dcadm.h>
 60  /*
 70   * XATMI stub header file
 80   */
 90  #include "rrspp_stbx.h"
100  main()
110  {
120  /*
130   * Define variables
140   */
150     int rc;
160  /*
170   * RPC-OPEN (start the UAP)
180   */
190      rc = dc_rpc_open(DCNOFLAGS);
200      if(rc != DC_OK){
210          printf("dc_rpc_open failed.\
220                  ERROR CODE = %d \n", rc);
230          goto PROG_END;
240      }
250  /*
260   * RPC-MAINLOOP (start the SPP service)
270   */
280      rc = dc_rpc_mainloop(DCNOFLAGS);
290      if(rc != DC_OK){
300          printf("dc_rpc_mainloop failed.\
310                  ERROR CODE = %d \n", rc);
320      }
330  /*
340   * RPC-CLOSE (terminate the UAP)
```

```
350   */
360  PROG_END:
370      dc_rpc_close(DCNOFLAGS);
380      exit(0);
390  }
```

- SPP coding sample (service function)

  The following shows a coding example (service function) of the SPP that was presented in the example of the request/response service.

```
 10  /* Example of service function of SPP (rrsvc.c file) */
 20  #include <stdio.h>
 30  #include <dcrpc.h>
 40  #include <xatmi.h>
 50  #include <dcadm.h>
 60  /*
 70   * XATMI stub header file
 80   */
 90  #include "rrspp_stbx.h"
100  void shotel(svcinfo)
110  TPSVCINFO *svcinfo;
120  {
130  /*
140   * Define variables
150   */
160      struct hotel *hptr;
170
180      hptr = (struct hotel *) svcinfo->data;
190      /* This service searches availability and returns
195       * status = 1 if no room is available,
200       * status = 0 if rooms are available,
205       * and a message if an error occurs.
210       * This example assumes that no room
215       * is available. */
220      hptr->status = 1 ;
230      tpreturn(TPSUCCESS, 0, hptr, 0, 0);
240      return ; /* In OpenTP1, return must be issued */
245               /* after tpreturn. */
250  }
260  void splane(svcinfo)
270  TPSVCINFO *svcinfo;
280  {
290      struct plane *pptr;
300      pptr = (struct plane *) svcinfo->data;
310      /* This service searches availability and returns
315       * status = 1 if no seat is available, status = 0
320       * if seats are available, and a message if an
325       * error occurs. This example assumes that no seat
```

```
330       * is available.
335       */
340      pptr->status = 1 ;
350      tpreturn(TPSUCCESS, 0, pptr, 0, 0);
360      return ;
370  }
```

- User service definition sample

  The following shows an example of user service definition of the SPP that was presented in the example of the request/response service.

```
10  # Example of user service definition (rrspp file)
20  set    service_group       = "rrspp_svg"
30  set    module              = "rrspp"
40  set    service             = "SVHOTEL=shotel",
45                               "SVPLANE=splane"
50  set    trn_expiration_time = 180
60  set    trn_expiration_time_suspend = Y
70  set    server_type = "xatmi"
```

(2) Conversational service paradigm sample

(a) Outline of processing

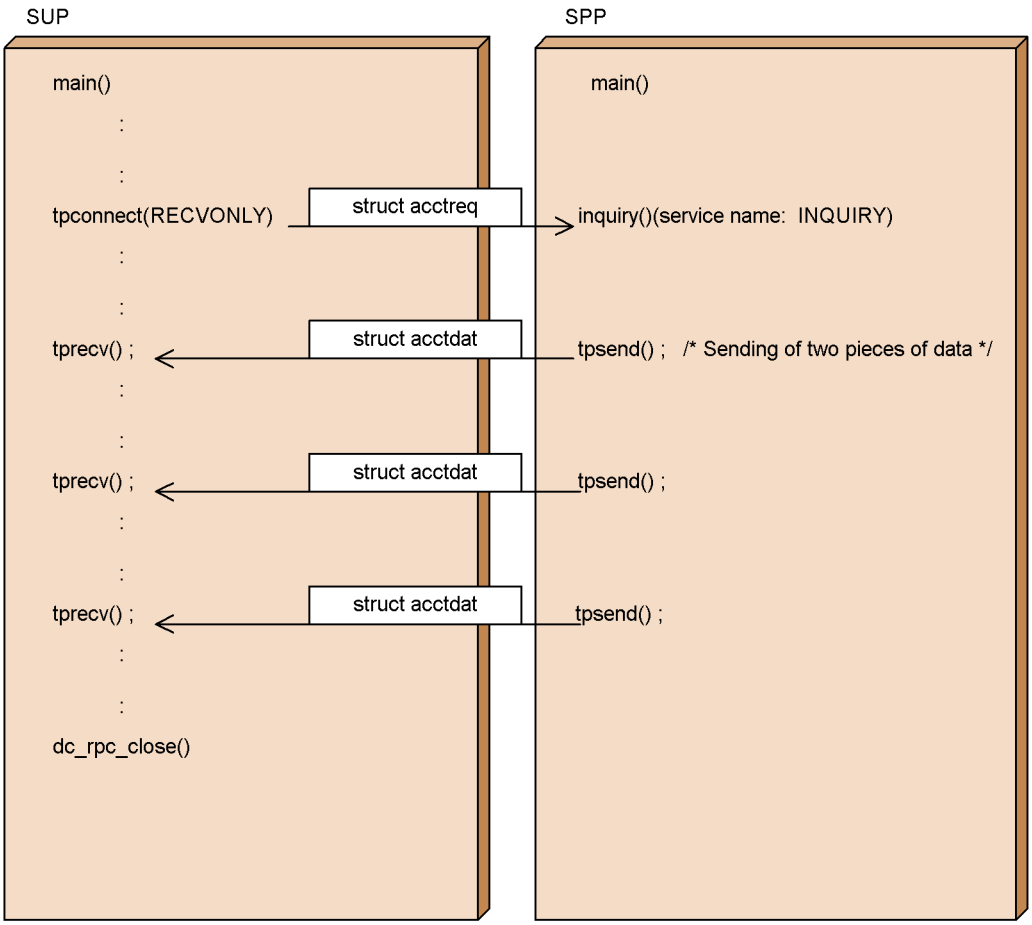
The processing of the sample here is outlined below.

The service function is activated through a typed buffer having a build of structure acctreq. The members of acctreq indicate the upper and lower limits of the account numbers. The service function sets account data in this range in the typed buffer having a build of structure acctdat and sends the data to the originator of the conversation.

(b) UAP configuration

The figure below shows the configuration of the sample UAP.

Figure 7-5: Communication of conversational service



(c) Typed buffers used

The structures of typed buffers used are shown below.

- Data for activating the service function

```
struct acctreq{
 long upper_no;
 long lower_no;
}
```

- Data for communication with the conversational service

```
struct acctdat{
 long acct_no;
 short amount;
 char name[128];
}
```

(d) SUP sample

- XATMI interface definition sample

  The following shows the XATMI interface definition of the SUP for the sample conversational service.

```
10  /* Example of XATMI interface definition of SUP */
15  /* (convsup.def file) */
20  called_servers = { "convspp.def" };
```

- SUP coding sample

  The following shows a coding example for the SUP used in the example of the conversational service.

```
10  /* Coding example of SUP (convsup.c file) */
20  #include <stdio.h>
30  #include <dcrpc.h>
40  #include <xatmi.h>
50  #include <tx.h>
60  #include <dcadm.h>
70  /*
80   * XATMI stub header file
90   */
100 #include "convsup_stbx.h"
110 main()
120 {
130 /*
140  * Define variables
150  */
160     struct   acctreq  *rptr;
170     struct   acctdata *dptr;
180     long     wlen;
190     int      cd;
200     int      rc;
210     long     revent;
220     long     size = 0 ;
230 /*
240  * RPC-OPEN (start the UAP)
250  */
260     rc = dc_rpc_open(DCNOFLAGS);
270     if(rc != DC_OK){
280         printf("dc_rpc_open failed. ERROR CODE = %d \n",
285                 rc);
```

```
290          goto PROG_END;
300      }
310   /*
320    * ADM-COMPLETE (report completion of user
       *               server start processing)
330    */
340      rc = dc_adm_complete(DCNOFLAGS);
350      if(rc != DC_OK){
360          printf("dc_adm_complete failed. ERROR "
365                 "CODE = %d \n", rc);
370          goto PROG_END;
380      }
390   /*
400    * TPALLOC (allocate typed buffer)
410    */
420     /* For setting minimum and maximum account numbers */
425     /* to be searched */
430      rptr = (struct acctreq *)tpalloc(X_COMMON,
435                                       "acctreq", 0);
440
450      if(rptr == NULL){
460          printf("tpalloc failed. ERROR CODE = %d \n",
465                  tperrno);
470          goto PROG_END;
480      }
490     /* For account data in the search result */
500      dptr = (struct acctdata *)tpalloc(X_COMMON,
505                                        "acctdata", 0) ;
510      if(dptr == NULL){
520          printf("tpalloc failed. ERROR CODE = %d \n",
525                  tperrno);
530          goto PROG_END;
540      }
550   /*
560    * Set data
570    * Specify the search range
580    */
590      rptr->lower_no = 10000000L;
600      rptr->upper_no = 20000000L;
610     /* Start the transaction  */
620      tx_begin() ;
630   /*
640    * TPCONNECT (call the conversational service)
650    * Call INQUIRY
660    */
670      cd = tpconnect("INQUIRY", (char *) rptr, 0,
675                     TPRECVONLY);
680      if(cd == -1){
```

```
690          printf("tpconnect failed. ERROR CODE = %d \n",
695                  tperrno);
700          goto PROG_END;
710      }
720   /*
730    * TPRECV (receive messages)
740    * Until an error occurs (include events),
750    */
760      while(rc != -1){
770          rc = tprecv(cd, (char **) &dptr, &wlen, 0,
775                      &revent);
780          /*
790           * If no error has occurred,
800           * output the received account information.
810           */
820          if(rc != -1) {
830              printf("The account information was "
835                     "received from the service.\n");
840              printf("Account number = %d \n",
845                      dptr->acct_no);
850              printf("Name = %s \n", dptr->name);
860              printf("Amount = %d \n", dptr->amount);
870          }
880      }
890   /*
900    * Output the result of the service
910    */
920      if(tperrno == TPEEVENT){
930          if(revent == TPEV_SVCSUCC){
940          /* The service was successful. */
950              printf("The service was successful.\n");
960              /* Transaction commit */
970              tx_commit() ;
980          }else{
990              printf("Some event has occurred. "
995                     "revent = %d\n",
1000             revent);
1010             /* Transaction rollback */
1020             tx_rollback() ;
1030         }
1040     }
1050  /*
1060   * Release the typed buffer
1070   */
1080     tpfree((char *) rptr);
1090     tpfree((char *) dptr);
1100  /*
1110   * RPC-CLOSE (terminate the UAP)
```

```
1120   */
1130      PROG_END:
1140      dc_rpc_close(DCNOFLAGS);
1150      exit(0);
1160   }
```

- User service definition sample

  The following shows an example of a user service definition of the SUP that was presented in the example of the conversational service.

```
 10  # Example of user service definition
 15  # (convsup file)
 20  set module        = "convsup"
 25                     # Name of executable file
 30  set watch_time     = 180
 35                     # Maximum time to wait for a response
 40  set receive_from  = none
 45                     # Receiving method
 50  set trn_expiration_time = 180
 60                     # Expiry time in transaction branch
 70  set trn_expiration_time_suspend = Y
 75                                  # Always specify Y
```

(e) SPP sample

- XATMI interface definition sample

  The following shows an example of XATMI interface definition of the SPP that was presented in the example of the conversational service.

```
 10  /* Example of XATMI interface definition of SPP */
 15  /* (convspp.def file) */
 20  X_COMMON acctreq {
 30      long   upper_no;
 40      long   lower_no;
 50  };
 60  X_COMMON acctdata {
 70      long   acct_no;
 80      char   name[128];
 90      short  amount;
100  };
110  service inquiry(X_COMMON acctreq) ;
```

- SPP coding sample (main function)

  The following shows a coding example (main function) of the SPP that was presented in the example of the conversational service.

```
 10  /* Example of SPP main function (convspp.c file) */
 20  #include <stdio.h>
 30  #include <dcrpc.h>
```

```
 40  #include <xatmi.h>
 50  #include <dcadm.h>
 60  /*
 70   * XATMI stub header file
 80   */
 90  #include "convspp_stbx.h"
100  main()
110  {
120  /*
130   * Define variables
140   */
150      int rc;
160  /*
170   * RPC-OPEN (start the UAP)
180   */
190      rc = dc_rpc_open(DCNOFLAGS);
200      if(rc != DC_OK){
210          printf("dc_rpc_open failed. ERROR "
215                 "CODE = %d \n", rc);
220          goto PROG_END;
230      }
240
250  /*
260   * RPC-MAINLOOP (start the SPP service)
270   */
280      rc = dc_rpc_mainloop(DCNOFLAGS);
290      if(rc != DC_OK){
300          printf("dc_rpc_mainloop failed. "
315                 "ERROR CODE = %d \n",rc);
310      }
320  /*
330   * RPC-CLOSE (terminate the UAP)
340   */
350      PROG_END:
360      dc_rpc_close(DCNOFLAGS);
370      exit(0);
380  }
```

- SPP coding sample (service function)

  The following shows a coding example (service function) of the SPP that was presented in the example of the conversational service.

```
 10  /* Example of service function of SPP */
 15  /* (convsvc.c file) */
 20  #include <stdio.h>
 30  #include <dcrpc.h>
 40  #include <xatmi.h>
 50  #include <dcadm.h>
```

```
 60  /*
 70   * XATMI stub header file
 80   */
 90  #include "convspp_stbx.h"
100  /*
110   * DEPOSITSVC service function
120   * Use tpconnect() to receive the minimum and maximum
125   * account numbers, and send information about
130   * accounts that are within that range
140   */
150  void inquiry(svcinfo)
160  TPSVCINFO *svcinfo;
170  {
180  /*
190   * Define variables
200   */
210      struct   acctreq  *rptr;
220      struct   acctdata *dptr;
230      char     type[9];
240      char     subtype[17];
250      long     revent, rval;
260      int      size;
270  /*
280   * Service request was accepted
290   */
300      rptr = (struct acctreq *) svcinfo->data;
310  /*
320   * Allocate the typed buffer for data that is to
325   * be returned to the originator
330   */
340      dptr = (struct acctdata *)tpalloc("X_COMMON",
345                                        "acctdata", 0);
350      if(rptr == NULL){
360          printf("An error occurred in tpalloc. "
365                 "tperrno = %d \n",
370                 tperrno);
380          abort();
390      }
400  /*
410   * User processing
420   * Search the data file and return the account
425   * information for account numbers within the specific
430   * range. This example assumes that two accounts have
435   * been found and then sends the data.
440   */
450
460      dptr->acct_no = 10000001L;
470      strcpy(dptr->name, "Hitachi Hanako");
```

```
480      dptr->amount = 20000;
490  /*
500   * TPSEND (send a message)
510   */
520      tpsend(svcinfo->cd, (char *) dptr, 0, 0, &revent);
530      if(tperrno != -1){
540          rval = TPSUCCESS;
550      }else{
560          rval = TPFAIL;
570          goto SVC_END;
580      }
590      dptr->acct_no = 10000002L;
600      dptr->amount = 10000;
610      strcpy(dptr->name, "Hitachi Tarou");
620  /*
630   * TPSEND (send a message)
640   */
650      tpsend(svcinfo->cd, (char *) dptr, 0, 0, &revent);
660      if(tperrno != -1){
670          rval = TPSUCCESS;
680      }else{
690          rval = TPFAIL;
700          goto SVC_END;
710      }
720  SVC_END:
730      tpreturn(rval, 0, NULL, 0, 0);
740      return; /* In OpenTP1, return is required after */
745              /* tpreturn. */
750  }
```

- User service definition sample

  The following shows an example of a user service definition of the SPP that was presented in the example of the conversational service.

```
 10  # Example of user service definition (convspp file)
 20    set  service_group     = "convspp_svg"
 25                           # Service group name
 30    set  module            = "convspp"
 35                           # Name of executable file
 40    set  service  = "INQUIRY=inquiry"
 50                  # Service name = entry point name
 60    set  watch_time        = 180
 65                  # Maximum time to wait for a response
 70    set  trn_expiration_time = 240
 80                  # Expiry time in transaction branch
 90    set  trn_expiration_time_suspend = Y
 95                                     # Always specify Y
100    set  server_type = "xatmi"       # Server type
```

```
110    set  receive_from = "socket"     # Receiving method
```

7.4.2 TX interface sample

This subsection shows a coding example for an SUP that uses the X/Open TX interface. This SUP uses TX-interfaced transaction control for processing that was described in *7.1 Coding samples for client/server configuration UAPs (SUP, SPP DAM access)*. See *7.1 Coding samples for client/server configuration UAPs (SUP, SPP DAM access)* for the process configuration and details of the SPP to which the service request is addressed.

```
 10
 20  /*
 30   * SUP01
 40   */
 50  #include <stdio.h>
 60  #include <string.h>
 70  #include <dcrpc.h>
 80  #include <tx.h>
 90
100  main()
110  {
120  /*
130   * Define variables
140   */
150      static char in_buf [1024];
160      static long in_buf_len;
170      static char out_buf [1024];
180      static long out_buf_len;
190      int rc;
200    TRANSACTION_TIMEOUT trn_timeout = 180;
205                     /* Monitoring interval 180 seconds */
210    TXINFO info;
220  /*
230   * RPC-OPEN (start the UAP)
240   */
250      rc = dc_rpc_open(DCNOFLAGS);
260      if(rc != DC_OK){
270          printf("SUP01:dc_rpc_open failed. "
275                 "CODE = %d \n",rc);
280          goto PROG_END;
290      }
300  /*
310   * TX-OPEN (open the resource manager)
320   */
330      rc = tx_open();
340      if(rc != TX_OK){
350          printf("SUP01:tx_open failed. CODE = %d \n",rc);
360          goto PROG_END;
```

```
370      }
380   /*
390    * TX-SET-TRANSACTION-TIMEOUT (set the transaction
395    *                             monitoring interval)
400    */
410      rc = tx_set_transaction_timeout(trn_timeout);
420      if(rc != TX_OK){
430          printf("SUP01:tx_set_transaction_timeout "
435                 "failed. CODE = %d \n",rc);
440          goto PROG_END;
450      }
460   /*
470    * ADM-COMPLETE (report completion of user server
475    * start processing)
480    */
490      rc = dc_adm_complete(DCNOFLAGS);
500      if(rc != DC_OK){
510          printf(dc_adm_complete failed. CODE = %d \n",
515                 rc);
520          goto PROG_END;
530      }
540   /*
550    * TX-BEGIN (start the transaction)
560    */
570      rc = tx_begin();
580      if(rc != TX_OK){
590          printf("SUP01:tx_begin failed. CODE = %d \n",
595                  rc);
600          goto TRAN_END;
610      }
620
630   /*
640    * TX-INFO (acquire transaction information)
650    */
660      rc = tx_info(&info);
670      if(rc <= 0){
680          printf("SUP01:Currently the system is not in "
685          "the transaction mode. CODE = %d \n",rc);
690          goto PROG_END;
700      }else if (rc == 1){
710      printf("SUP01:return=%d,control=%d,timeout=%d,"
715             "state=%d\n",
720        info.when_return,info.transaction_control,
730        info.transaction_timeout, info.transaction_state);
740     }
750   /*
760    * RPC-CALL (request a remote service)
770    */
```

```
 780      strcpy(in_buf,"SUP01:DATA OpenTP1!!");
 790      in_buf_len = strlen(in_buf) + 1;
 800      out_buf_len = 1024;
 810      rc = dc_rpc_call("svr01","svr01",in_buf,&in_buf_len,
 820      out_buf,&out_buf_len,DCNOFLAGS);
 830      if(rc != DC_OK){
 840          printf("SUP01:The service request failed. "
 845                 "CODE = %d \n",rc);
 850          goto TRAN_END;
 860      }
 870      printf("SUP01:SERVICE FUNCTION RETURN = %s\n",
 875              out_buf);
 880  /*
 890   * TX-SET-TRANSACTION-CONTROL (set the unchained mode)
 900   */
 910      TRAN_END:
 920      rc = tx_set_transaction_control(TX_UNCHAINED);
 930      if(rc != TX_OK){
 940          printf("SUP01:tx_set_transaction_control "
 945                 "failed. CODE = %d \n",rc);
 950      }
 960  /*
 970   * TX-COMMIT (commit in unchained mode)
 980   */
 990      rc = tx_commit();
1000      if(rc != TX_OK){
1010          printf("SUP01:tx_commit failed. CODE = %d \n",
1015                  rc);
1020      }
1030  /*
1040   * TX-CLOSE (close the resource manager)
1050   */
1060      PROG_END:
1070      rc = tx_close();
1080      if(rc != TX_OK){
1090          printf("SUP01:tx_close failed. CODE = %d \n",
1095                  rc);
1100          goto PROG_END;
1110      }
1120  /*
1130   * RPC-CLOSE (terminate the UAP)
1140   */
1150      dc_rpc_close(DCNOFLAGS);
1160      printf("SUP01:Processing is finished.\n");
1170      exit(0);
1180  }
```

7.5 TxRPC examples (templates created by the IDL compiler)

This section explains templates output by the IDL compiler. The user should modify these templates depending on the work.

7.5.1 Outline of creation procedures

This subsection outlines the creation procedures.

(1) Creating a stub and coding a UAP

The procedures for creating a stub and coding a UAP are explained below.

(a) For IDL-only TxRPC

1. Create the following files:

   (1) IDL file

   (2) Client program

   (3) Manager program

2. Use the `txidl` compiler to compile the IDL file that was created in step 1. As a result, the files below are created. A value enclosed in parentheses indicates a default name. (*xxxx* indicates an IDL file name.)

   (4) Template of a server program (The name is fixed to `serv.c`)

   The template of a server program can be used without modification. To change the contents, change the name as required, then code the additional processing.

   (5) Template of a user service definition

   The template of a user service definition cannot be used without modification. For how to define the required items, see the explanations of the user service definition in the manual *OpenTP1 System Definition*.

   (6) Template of an environment definition file (when the `-cptype wdce` option is specified)

   The template of an environment definition file cannot be used without modification. For how to define the required items, see the explanations of the client environment definition in the manual *OpenTP1 TP1/Client/W, TP1/Client/P*.

   (7) Client stub (*xxxx*`_cstub.c`)

   (8) Server stub (*xxxx*`_sstub.c`)

   (9) Header file (*xxxx*`.h`)

(2) Compiling and linking UAPs

Compile the program with the C compiler. The library to be linked depends on the specified process type. The libraries to be linked are as follows:

`-lbetran`

TP1/Server Base library

`-lclt`

TP1/Client library

The process types to be specified and the libraries required for the client and server programs are listed below.

- `-cptype ndce` and `-sptype ndce`

  Client: `-lbetran` and `-ltactk`

  Server: `-lbetran` and `-ltactk`

- `-cptype wdce` and `-sptype ndce`

  Client: `-ltpldce` and `-lclt`, and DCE-related libraries

  Server: `-lbetran` and `-ltactk`

- `-cptype ndce` and `-sptype wdce`

  Client: `-lbetran` and `-ltactk`

  Server: `-ltpldce`, `-lbetran`, and DCE-related libraries

- `-cptype wdce` and `-sptype nbet`

  Client: DCE-related libraries

  Server: DCE-related libraries

- `-cptype nbet` and `-sptype wdce`

  Client: DCE-related libraries

  Server: DCE-related libraries

- `-cptype nbet` and `-sptype nbet`

  Client: DCE-related libraries

  Server: DCE-related libraries

7.5.2 Examples of Files

This subsection gives examples of the following files:

- IDL file

- Client program
- Manager program
- ACF file
- Template of a server program
- Template of a user service definition
- Template of an environment definition

(1) Example of an IDL file

The following shows an example of an IDL file.

```
 10   /*
 20    *  (1) Example of IDL file (sample.idl)
 30    */
 40   [
 50   uuid(f990a82a-10e5-11ce-9b02-0000870000ff),
 60   version(1.0),
 70   transaction_mandatory
 80   ]
 90   interface sample_ope
100   {
110     const  long  NAME_LENGTH = 20;
115                   /* size of name field in record */
120     const  long  AGE_LEN = 3;
125                   /* size of age field in record */
130     const  long  MAXRECORD = 10;
135                   /* max number of records in database */
140
150     /* struct info:                                    */
155     /* record format of customer information database */
160     typedef struct  info{
170       char  name[NAME_LENGTH];  /* name (20 bytes) */
180       char  sex;                /* sex (1 byte) */
190       char  age[AGE_LEN];       /* age (3 bytes) */
200       long  sale;               /* sales (4 bytes) */
210     }info_t;
220
230      error_status_t  getinfo
240     (
250       [in] unsigned   char  name[NAME_LENGTH],
255                             /* input parameter */
260       [out] info_t    *ptr    /* output parameter */
270     );
280   }
290   /* EOF */
```

(2) Example of a client program

The following shows an example of a client program.

```
  10   /*
  20    *  (2) Example of a client program
  30    *  Note: dc_rpc_open(), dc_adm_complete(),
  40    *        and dc_rpc_close() are required for
  45    *        the ndce type;
  50    *        dc_clt_cltin(), dc_rpc_open(),
  55    *        dc_rpc_close(), and dc_clt_cltin()
  60    *        are required for the wdce type.
  70    *        For the header file to be included,
  75    *        use the TP1/Client library.
  80    *  clt.c
  90    *  Functions = main()
 100    */
 110
 120   #include <stdio.h>
 130   #include <dcrpc.h>
 140   #include <dctrp.h>
 150   #include <dcadm.h>
 160   #include <tx.h>
 170   #include "sample.h"
 180
 190   /*
 200    *  Program Specification
 210    *  Build customer information database.
 215    *  Allow actions noting the following.
 220    *   * Reference processing
 230    *     Refer to information using "name" as the key.
 240    *
 250    *  Customer information database
 260    *  *-----------------------------------------*
 270    *  | Name     | Sex    | Age  | Sales    |
 280    *  |-----------------------------------|
 290    *  | Smith    | Male   | 30   | 10,000   |
 300    *  | Johnson  | Female | 23   | 15,000   |
 310    *  | Williams | Female | 26   |  8,000   |
 320    *  | Jones    | Male   | 24   | 10,000   |
 330    *  | Brown    | Male   | 35   | 18,000   |
 340    *  | Davis    | Male   | 20   |  3,000   |
 350    *  | Miller   | Female | 28   | 10,000   |
 360    *  | Wilson   | Female | 27   | 21,000   |
 370    *  | Moore    | Male   | 25   |  6,000   |
 380    *  | Taylor   | Male   | 24   | 11,000   |
 390    *  *-----------------------------------------*
 400    *
 410    *  This program requires service.
```

```
420    *    <refer>  refer Taylor's information.
430    *
440    */
450   /*
460    *  name = main()
470    *  func = Client program for sample_ope interface
480    *       (1) service requirement (reference)
490    *       (2) output result of service requirement
500    *  arg = nothing
510    *  return = void
520    */
530
540   int  main()
550   {
560     static unsigned char name[] = "Taylor";
565                                /* input parameter */
570     info_t out_data;           /* output parameter */
580     error_status_t status;     /* return code for server */
590     int     rc;                /* return code */
600
610   /*
620    *  Start UAP
630    */
640     rc = dc_rpc_open(DCNOFLAGS);
650     /* error processing */
660     if(rc != DC_OK){
670       fprintf(stderr,"client:dc_rpc_open failed. "
675                      "rc = %d\n",rc);
680       goto END;
690     }
700
710     /*
720     **  Post completion of user process start processing
730     */
740     rc = dc_adm_complete(DCNOFLAGS);
750     /* error processing */
760     if(rc != DC_OK){
770       fprintf(stderr,"client:dc_adm_complete failed. "
775                      "rc = %d\n",rc);
780       goto END;
790     }
800
810   /*
820    *  Begin  transaction
830    */
840
850     rc = tx_begin();
860     /* error processing */
```

```
  if(rc != DC_OK){
    fprintf(stderr,"client:tx_begin failed. "
                   "rc = %d\n",rc);
    goto END;
  }

/*
 *  getinfo:
 *  get information for input parameter
 */
  status = getinfo(name,&out_data);
  if(status != 0){
    fprintf(stderr,"client:getinfo "
                   "failed.rc = %d\n",status);
  }else{
    fprintf(stdout,"NAME: %s SEX: %c AGE: %s "
                   "SALE:%ld\n",
        out_data.name,
        out_data.sex,
        out_data.age,
        out_data.sale);
  }
/*
 *  commit  transaction
 */

  rc = tx_commit();
  /* error processing */
  if(rc != DC_OK){
    fprintf(stderr,"client:tx_commit failed. "
                   "rc = %d\n",rc);
    goto END;
  }
/*
 *  Termination processing
 */
END:
  dc_rpc_close(DCNOFLAGS);
  return(0);
}
```

(3) Example of a manager program

The following shows an example of a manager program.

```
/*
 *
 *    (3) Example of manager program
 *  sv.c
 *  Data Table = customers
```

```
 60    *  Functions = main()
 70    *        getinfo()
 80    */
 90
100   #include <stdio.h>
110   #include <string.h>
120   #include "sample.h"
130
140   /*
150    *  name   = customers
160    *  func   = customer information database
170    *  field  = name (20 bytes)
180    *      sex (1 byte)
190    *      age (3 bytes)
200    *      sales (4 bytes)
210    *  record  = 10 records (1 record = 28 bytes)
220    */
230   static info_t customers[MAXRECORD] =
240              { {"Smith",   'M',"30",10000},
250                {"Johnson", 'F',"23",15000},
260                {"Williams",'F',"26", 8000},
270                {"Jones",   'M',"24",10000},
280                {"Brown",   'M',"35",18000},
290                {"Davis",   'M',"20", 3000},
300                {"Miller",  'F',"28",10000},
310                {"Wilson",  'F',"27",21000},
320                {"Moore",   'M',"25", 6000},
330                {"Taylor",  'M',"24",11000}
340             };
350
360   /*
370    *  name = getinfo()
380    *  func = Manager routine for sample_ope interface
390    *    (1) search suitable record.
400    *    (2) set found record to output parameter.
410    *  arg =   name   :i: name
420    *      out_data:o: information for input parameter
430    *  return = result
440    *      0 : success getinfo
450    */
460
470   error_status_t  getinfo(name,out_data)
480   unsigned char  *name;
490   info_t  *out_data;
500   {
510     int i;          /* counter of for loop */
520     info_t *ptr;    /* pointer for search record */
530
```

```
540   /* point 1st record of database(customers) */
550     ptr = customers;
560
570   /* search until record found with same name */
575   /* or end of database                       */
580     for (i = 0; i < MAXRECORD; i++, ptr++) {
590       /* compare name */
600       if(strcmp(name,ptr->name) == 0) {
610         memcpy(out_data,ptr,sizeof(info_t));
620         return (0);
630         }
640     }
650     return(1);
660   }
```

(4) Example of an ACF file

The following shows an example of an ACF file.

```
 10   /*
 20    *
 30    *  (4) Available only in the example of
 40    *      ACF file RPC TxRPC sample.acf
 50    */
 60
 70   [auto_handle] interface sample_ope
 80   {
 90     [comm_status, fault_status] getinfo();
100   }
```

(5) Template example of a server program

The template example of a server program depends on the value specified for the argument of the `txidl` command. The following shows an example when the option `specified.-sptype ndce` is specified.

```
 10   /*
 20    *
 30    *    (5) Template for server program (name: serv.c)
 40    *    <For -sptype ndce>
 50    */
 60
 70   #include <dctrp.h>
 80
 90   main()
100   {
110     idl_long_int rc;
120     rc = dc_rpc_open(DCNOFLAGS);
130     if(rc != DC_OK) {
140       printf("server : dc_rpc_open failed. rc=%d\n", rc);
150       goto end_of_program;
```

```
160     }
170     rc=dc_rpc_mainloop(DCNOFLAGS);
180     if(rc != DC_OK) {
190       printf("server : dc_rpc_mainloop failed. "
195              "rc=%d\n", rc);
200     }
210   end_of_program:
220     dc_rpc_close(DCNOFLAGS);
230     exit(0);
240   }
```

(6) Template example of a user service definition

The template example of a user service definition depends on the value specified for the argument of the `txidl` command. The following shows an example when each option is specified.

- When the `-cptype ndce` option is specified

```
10   /*
20    *  (6) Example of user service definition template
30    *      <For -cptype ndce>
40    */
50
60   #Don't change the 2 definitions below.
70
80   set atomic_update = Y
90
100  set trn_expiration_time_suspend = Y
110
120  # If this program is SUP, set none.
125  # If other, set queue or socket.
130
140  set receive_from = none
150
160  #Set your modulename.
170
180  set module = "modulename"
190
200  #Set non-zero value.
210
220  set trn_expiration_time = 180
230
240  #Add any definition you need.
```

- When the `-sptype ndce` option is specified

```
10   /*
20    *  (6) Example of user service definition template
30    *      <For -sptype ndce>
40    */
```

```
 50
 60   #Don't change the 4 definitions below.
 70
 80
 90   set atomic_update = Y
100
110   set trn_expiration_time_suspend = Y
120
130   set service_group = "sample_ope"
140
150   set service = "_getinfo=_getinfo"
160
170   #Set your modulename.
180
190   set module = "modulename"
200
210   #Set non-zero value.
220
230   set trn_expiration_time = 180
240
250   #Add any definition you need.
```

- When the `-sptype wdce` option is specified

```
 10   /*
 20    *  (6) Example of user service definition template
 30    *      <For -sptype wdce>
 40    */
 50
 60   #Don't change the  4 definitions below.
 70
 80   set atomic_update = N
 90
100   set receive_from = queue
110
120   set service_group = "sample_ope"
130
140   set service = "_getinfo=_getinfo"
150
160   #Set your modulename.
170
180   set module = "modulename"
190
200   #Add any definition you need.
```

(7) Template example of an environment definition

The following shows an example of an environment definition template.

```
 10   /*
 20    *  (7) Example of an environment definition template
```

```
 30    *       <For -cptype wdce>
 40    */
 50
 60   #Set the 2 definitions below
 70
 80   #DCNAMPORT =
 90
100   #DCHOST =
110
120   #Add any definition you need.
```

Chapter

8. Reference for Application Activation

This chapter explains user exit routines and MCF event reference information which are related to the facility for activating application programs in an environment where the TP1/Message Control is used.

This chapter contains the following sections:

Function format of the user exit routine that determines whether to inherit the timer-start settings

Structure format of mcf event that reports discarding of a timer-start message (ERREVT4)

Function format of the user exit routine that determines whether to inherit the timer-start settings

The exit routine for determining timer start inheritance is called in the following format:

Format

- ANSI C , C++

```
#include <dcmpsv.h>
DCLONG  uoc_func(dcmpsv_uoc_rtime  *parm)
```

- K&R C

```
#include <dcmpsv.h>
DCLONG  uoc_func(parm)
dcmpsv_uoc_rtime  *parm;
```

Description

If the timer-started function `dc_mcf_execap()` is followed by an error which raises the need for rerunning the OpenTP1, this exit routine can change the timer-start environment. It can perform the following:

- Inherit or cancel the current timer-start
- Make inherited timer-start immediate start
- Change the name of the application to be timer-started

When installing in the MCF the exit routine that determines the inheriting timer-start message, specify the address of the exit routine function in the MCF main function for the application startup service. The MCF main function for the application startup service does not depend on the communication protocol.

For details on how to create the MCF main function for the application startup service, see the manual *OpenTP1 Operation*.

When `uoc_func` (exit routine that determines the inheriting time-start message) is called, the following parameters are passed from the MCF to parm.

Parameters

- dcmpsv_uoc_rtime

```
typedef  struct {
     char  le_name[9];    ... Input source logical terminal name
     char  reserve1[7];   ... Reserved
     char  ap_name[9];    ... Application name
     char  reserve2[7];   ... Reserved
     DCLONG  exec_time;   ... Timer-start time
     char  ap_type;       ... Application type
                          'a': ans type; 'n': noans type
     char  time_type;     ... Timer-start type
                          'i': Interval specification for timer start
                          't': Time point specification for timer start
     char  reserve3[26]; ... Reserved
} dcmpsv_uoc_rtime;
```

Arguments whose value is passed from MCF to exit routine

- `le_name`

  The input source logical terminal name is set here. If the function `dc_mcf_execap()` is called from the SPP, '`*`' is set here.

- `ap_name`

  The application name specified by the UAP in the timer-started function `dc_mcf_execap()` is set here.

- `exec_time`

  The MHP start time specified by the UAP in the timer-started function `dc_mcf_execap()` is set here, as the number of seconds counted from 00:00:00 on January 1, 1970.

- `ap_type`

  The application type of the UAP which issued the timer-started function `dc_mcf_execap()` is set here:

  '`a`': ans type

  '`n`': noans type

- `time_type`

  The timer-start type specified by the UAP in the timer-started function `dc_mcf_execap()` is set here:

  '`i`': Interval specification for timer start

  '`t`': Time point specification for timer start

Arguments whose value is set in the exit routine

- ap_name

  To change the application to be timer-started, specify the new application name here. The name specified here has effect when DCMPSV_UOC_TIME_JUST is specified for the return value.

Return values

uoc_func() must return the following values:

Return value	Explanation
DCMPSV_UOC_TIME_CONTINUE	Timer-start is inherited.
DCMPSV_UOC_TIME_JUST	Immediate start will be in effect.
DCMPSV_UOC_TIME_DEQ	Timer-start is canceled.

The subsequent MCF processing varies depending on the return value from uoc_func() as follows:

- DCMPSV_UOC_TIME_CONTINUE

  If this value is returned from the exit routine, the MCF counts the seconds from 00:00:00 on January 1, 1970 to the present time and compares it with the time specified in the function dc_mcf_execap(). If the present time is later than the time specified in the function, the MCF immediately starts the pertinent MHP. Otherwise, the application will be timer-started.

- DCMPSV_UOC_TIME_JUST

  If this value is returned from the exit routine, the MCF immediately starts the pertinent MHP. If this value is to be returned, the application to be immediately started can be changed in the exit routine. However, change to an MHP for MCF event processing is not allowed. If the specified new application name is not defined, ERREVT4 is reported.

  If the application name of the UAP to be immediately started by the exit routine is changed and the application types of the old and new MHPs to be started are different, the segments to be timer-started are deleted from the output queue, with the output of a warning message (KFCA10711-W).

- DCMPSV_UOC_TIME_DEQ

  If this value is returned from the exit routine, the MCF cancels timer-start. The segments to be timer-started are deleted from the output queue, with the output of an information message (KFCA10700-I).

If another value is returned from the exit routine, the segments to be timer-started are

deleted from the output queue, with the output of a warning message (KFCA10710-W).

Notes on creating user exit routines

- Functions available to user exit routines

  When creating a user exit routine, you can use only the following functions in a user exit routine. Note that using any other function may prevent the user exit routine from operating normally.

  - Memory manipulation functions

    Data area management (example: malloc, free)

    Shared memory management (example: shmctl, shmget, shmop)

    Memory manipulation (example: memcpy)

    Character string manipulation (example: strcpy)

  - Time acquisition functions

- User exit routine errors

  When an error is detected in a user exit routine, report the error to the MCF using the return code prescribed by the MCF. If a process-terminating signal or abort() is issued in a user exit routine, the MCF terminates abnormally.

- User exit routine execution timing

  Execution timing of a user exit routine started by the MCF may not always synchronize with startup or termination sequence of the OpenTP1 system or UAP. Create user exit routines so that there is no problem if the user exit routine is executed before UAP or the user exit routine is called after all UAPs have terminated.

- Local variable size of user exit routines

  Design the local variables to be used in user exit routines so that the total size within each user exit routine does not exceed 1-kilobyte. In addition, do not issue a recursive call of a function within a user exit routine.

Structure format of mcf event that reports discarding of a timer-start message (ERREVT4)

The format of the structure passed as the first segment of the event that reports discarding of a timer-start message (ERREVT4) is shown below. This structure is defined in the header file <dcmcf.h>. Include the file <dcmcf.h> with the #include statement for the MHP which handles the MCF event information. For the format of MCF event information other than ERREVT4, see the explanation in the applicable *OpenTP1 Protocol* manual.

MCF event information common header

```
struct  dc_mcf_evtheader {
     char  mcfevt_name[9]; ... MCF event code
     char  le_name[16];    ... Input source logical terminal name
     char  cn_name[9];     ... Connection name
     unsigned  char  format_kind; ... Area used by the MCF
     char  reserve01;      ... Reserved
     DCLONG  time;         ... Message input time
} ;
```

ERREVT4 format

```
struct  dc_mcf_evt4_type {
     struct  dc_mcf_evtheader evtheader; ... MCF event common
                                             header
     char  reserve01[12]; ... Reserved
     char  reserve02[10]; ... Reserved
     char  reserve03[2]; ... Reserved
     char  ap_name[10]; ... Application name (corresponding to
                              timer-start message)
     short  reason_code; ... Reason code
} ;
```

Arguments

- le_name

  The name of the logical terminal where the message was input is set here. In the following cases, '*' is set here:

  - An error occurred in the MHP which was started by the function dc_mcf_execap() from the SPP.
  - In addition to the above error, another error occurred in the MHP which was started by the function dc_mcf_execap() from the MHP that was started as an MCF event processing MHP.

- `cn_name`

  The connection name is set here. In the following cases, '`*`' is set here:

  - An error occurred in the MHP which was started by issuing the function `dc_mcf_execap()` from the SPP.
  - In addition to the above error, another error occurred in the MHP which was started by issuing the function `dc_mcf_execap()` from the MHP that was started as an MCF event processing MHP.

- `time`

  The message input time is set here as the number of seconds counted from 00:00:00 on January 1, 1970.

- `ap_name`

  The name of the application which is specified in the timer-started function `dc_mcf_execap()` and encountered an error is set here.

- `reason_code`

  The `ERREVT4` reason code is set here. The reason codes are detailed below.

Reason code in C language (hexadecimal)	Reason
`DCMCF_SCD_ERR` (0020)	The MHP or SPP could not be activated because of an RPC error or inactive server.
`DCMCF_QUE_BUF_ERR` (0030)	Since memory became insufficient, data could not be written to the input queue.
`DCMCF_QUE_FIL_OVER` (0031)	Since the queue file is full, data could not be written to the input queue.
`DCMCF_QUE_LIMIT_OVER` (0032)	Since the maximum number of storable input messages exceeded the defined value, data could not be written to the input queue.
`DCMCF_QUE_IO_ERR` (0033)	An error occurred in writing to the input queue.
`DCMCF_AP_CLOSE` (0040)	An MHP application is being shut down.
`DCMCF_AP_SECURE` (0041)	An MHP application is in the secure status.
`DCMCF_SERV_CLOSE` (0042)	An MHP service or service group is being shut down.
`DCMCF_SERV_SECURE` (0043)	An MHP service group is in the secure status.

Appendix

A. Using OpenTP1 Remote Procedure Calls and XATMI-interfaced Functions in Combination

This appendix explains how to use OpenTP1 inter-process communication (OpenTP1 remote procedure calls and XATMI interface functions).

A.1 Modes of combined use

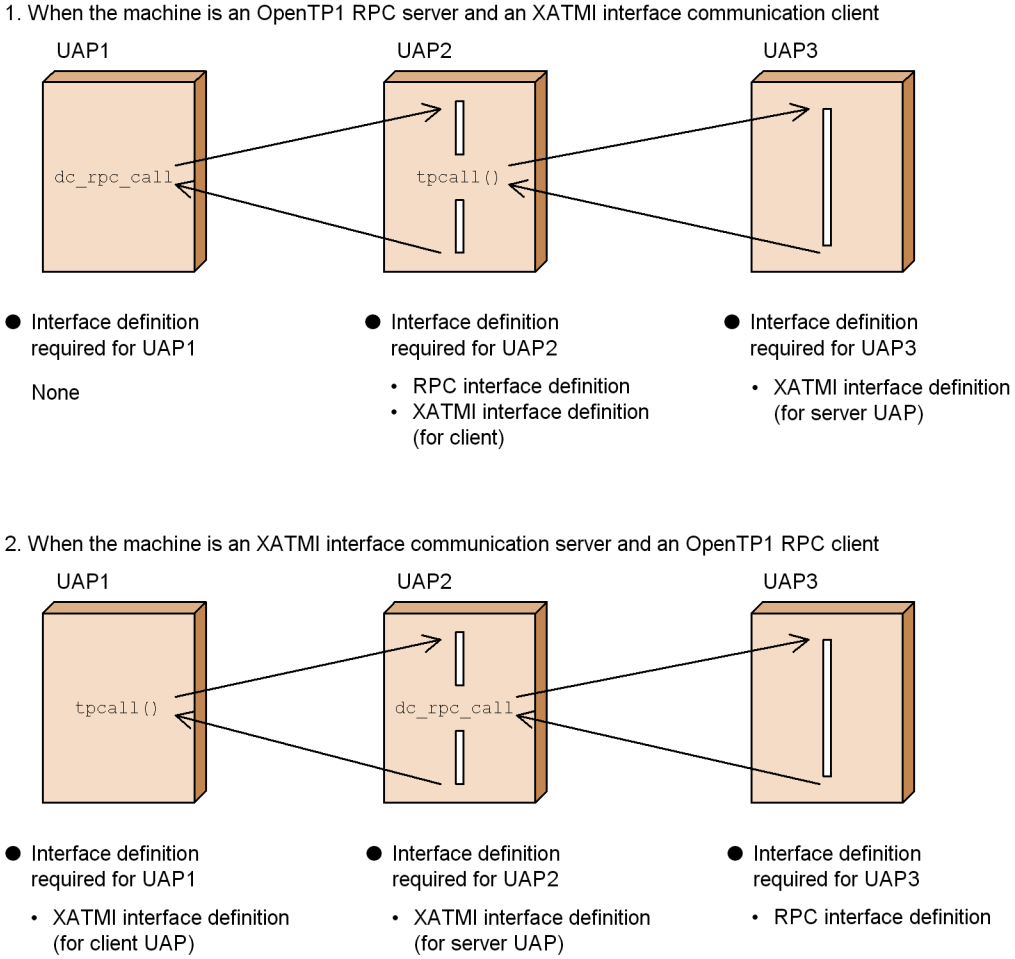
There are the following modes of combined use:

1. When the machine is an OpenTP1 RPC server and an XATMI interface communication client
2. When the machine is an XATMI interface communication server and an OpenTP1 RPC client

In mode (1), specify RPC and XATMI interface definitions for one file when creating a stub, and execute the `stbmake` command or `tpstbmk` command.

The figure below shows the modes of combined use of inter-process communication and the stubs required.

Figure A-1: Modes of combined use of inter-process communication and the stubs required



A.2 Creating stubs of application programs that are used together

This section explains how to create the stubs of UAPs that are called from the function `dc_rpc_call()` and call XATMI interface functions (such as `tpcall()`).

To create the UAP:

1. Create an interface definition file.

   For the file to be created, specify the RPC and XATMI interface definitions (for the client). Suffix the file name with `.def`.

2. Execute the `stbmake` command or `tpstbmk` command.

Specify the required arguments for the `stbmake` command, and execute the command. Execution of the command creates the declaration files listed below. *xxxxx* indicates a character string of an interface definition file name from which `.def` is excluded.

- OpenTP1 RPC stub source file (default file name: *xxxxx*`_sstb.c`)
- XATMI stub source file (default file name: *xxxxx*`_stbx.c`)
- XATMI stub header file (default file name: *xxxxx*`_stbx.h`)

If the RPC interface definition and XATMI interface definition coexist, the XATMI stub source file and XATMI stub header file are created.

3. Compile the stub source files and link them with a UAP.

   Compile the source files created in step 2 with the C compiler, and link them with a UAP.

A.3 Callable XATMI interface functions

The table below lists XATMI interface functions that can be used by an SPP called by the function `dc_rpc_call()`. The stubs explained in *A.2 Creating stubs of application programs that are used together* must have been linked with the SPP that called these functions.

Table A-1: XATMI interface functions that can be used by an SPP called by the function dc_rpc_call()

XATMI interface function	Call
`tpacall`	Y
`tpadvertise`	N
`tpalloc`	Y
`tpcall`	Y
`tpcancel`	Y
`tpconnect`	Y
`tpdiscon`	Y
`tpgetrply`	Y
`tpfree`	Y
`tprecv`	Y
`tprealloc`	Y
`tpreturn`	N

XATMI interface function	Call
tpsend	Y
tpservice	N
tptypes	Y
tpunadvertise	N

Legend:

Y: Can be called.

N: Cannot be called.

Note

The tpservice indicates the entity of the service function.

B. Changes to the Interfaces (for Migrating from Version 6 or Earlier)

If you migrate from Version 6 or earlier of TP1/Message Control and the architecture is not 32-bit, you must check the C language source files. This appendix lists the changes to the interfaces from when migrating from Version 6 or earlier.

The following table provides an overview by interface of the changes that are explained in this appendix.

Table B-1: List of changes to the interfaces

Changed interface		Reference in the Version 7 manual
Message transmission interfaces	`dc_mcf_ap_info`	*2. dc_mcf_ap_info*
	`dc_mcf_ap_info_uoc`	*2. dc_mcf_ap_info_uoc*
	`dc_mcf_close`	*2. dc_mcf_close*
	`dc_mcf_commit`	*2. dc_mcf_commit*
	`dc_mcf_contend`	*2. dc_mcf_contend*
	`dc_mcf_execap`	*2. dc_mcf_execap*
	`dc_mcf_mainloop`	*2. dc_mcf_mainloop*
	`dc_mcf_open`	*2. dc_mcf_open*
	`dc_mcf_receive`	*2. dc_mcf_receive*
	`dc_mcf_rollback`	*2. dc_mcf_rollback*
	`dc_mcf_tempget`	*2. dc_mcf_tempget*
	`dc_mcf_tempput`	*2. dc_mcf_tempput*
	`dc_mcf_timer_cancel`	*2. dc_mcf_timer_cancel*
	`dc_mcf_timer_set`	*2. dc_mcf_timer_set*
User exit routines	User exit routine that determines inheritance of timer-start messages	*8. Function format of the user exit routine that determines whether to inherit the timer-start settings*
MCF event interfaces		*8. Structure format of mcf event that reports discarding of a timer-start message (ERREVT4)*
Coding example for the MHP service function		*7.3 Coding samples for message exchange configuration UAPs (MHP)* *(2) MHP sample (service function)*

The following sections explain the changes to the interfaces between Version 6 or earlier and Version 7. Changes are indicated by underlines.

B.1 Message transmission interfaces

This section lists the changes to the message transmission interfaces.

(1) dc_mcf_ap_info - Report the application information

(a) ANSI C, C++

Version 6 or earlier	Version 7

<For 32-bit architecture>
`int dc_mcf_ap_info(long flags,`
`char *mcfid,`
`char *apname,`
`struct DC_MCFAPINFO *apinfo,`
`char *resv01,`
`long resv02)`
	`int dc_mcf_ap_info(DCLONG flags,`
	`char *mcfid,`
	`char *apname,`
	`struct DC_MCFAPINFO *apinfo,`
	`char *resv01,`
	`DCLONG resv02)`

<For 64-bit architecture>
`int dc_mcf_ap_info(int flags,`
`char *mcfid,`
`char *apname,`
`struct DC_MCFAPINFO *apinfo,`
`char *resv01,`
`int resv02)`	

(b) K&R C

Version 6 or earlier	Version 7

<For 32-bit architecture>
`int dc_mcf_ap_info(flags,`
`mcfid,`
`apname,`
`apinfo,`
`resv01,`
`resv02)`
`long flags;`
`char *mcfid;`
`char *apname;`
`struct DC_MCFAPINFO *apinfo;`
`char *resv01;`
`long resv02;`
	`int dc_mcf_ap_info(flags,`
	`mcfid,`
	`apname,`
	`apinfo,`
	`resv01,`
	`resv02)`
	`DCLONG flags;`
	`char *mcfid;`
	`char *apname;`
	`struct DC_MCFAPINFO *apinfo;`
	`char *resv01;`
	`DCLONG resv02;`

Version 6 or earlier	Version 7
<For 64-bit architecture> int dc_mcf_ap_info(flags, mcfid, apname, apinfo, resv01, resv02) int flags; char *mcfid; char *apname; struct DC_MCFAPINFO *apinfo; char *resv01; int resv02;	

(c) Arguments whose value is returned from OpenTP1

- **apinfo**

Version 6 or earlier	Version 7

<For 32-bit architecture>
struct DC_MCFAPINFO {
char mcf_apinfo[4];
long mcf_resv00;
char mcf_ap_name[9];
char mcf_ap_mcfid[3];
char mcf_resv01[4];
long mcf_ap_stat;
long mcf_ap_type;
char mcf_sg_name[32];
long mcf_sg_stat;
long mcf_sg_hold;
char mcf_sv_name[32];
long mcf_sv_stat;
long mcf_ap_ntmetim;
long mcf_ap_tempsize;
long mcf_ap_msgcnt;
long mcf_ap_trnmode;
long mcf_ap_quekind;
char mcf_resv02[72];
}
	struct DC_MCFAPINFO {
	char mcf_apinfo[4];
	DCLONG mcf_resv00;
	char mcf_ap_name[9];
	char mcf_ap_mcfid[3];
	char mcf_resv01[4];
	DCLONG mcf_ap_stat;
	DCLONG mcf_ap_type;
	char mcf_sg_name[32];
	DCLONG mcf_sg_stat;
	DCLONG mcf_sg_hold;
	char mcf_sv_name[32];
	DCLONG mcf_sv_stat;
	DCLONG mcf_ap_ntmetim;
	DCLONG mcf_ap_tempsize;
	DCLONG mcf_ap_msgcnt;
	DCLONG mcf_ap_trnmode;
	DCLONG mcf_ap_quekind;
	char mcf_resv02[72];
	}

Version 6 or earlier	Version 7

<For 64-bit architecture>
struct DC_MCFAPINFO {
char mcf_apinfo[4];
int mcf_resv00;
char mcf_ap_name[9];
char mcf_ap_mcfid[3];
char mcf_resv01[4];
int mcf_ap_stat;
int mcf_ap_type;
char mcf_sg_name[32];
int mcf_sg_stat;
int mcf_sg_hold;
char mcf_sv_name[32];
int mcf_sv_stat;
int mcf_ap_ntmetim;
int mcf_ap_tempsize;
int mcf_ap_msgcnt;
int mcf_ap_trnmode;
int mcf_ap_quekind;
char mcf_resv02[72];
}	

(2) dc_mcf_ap_info_uoc - Report application information to a user exit routine

(a) ANSI C, C++

Version 6 or earlier	Version 7
<For 32-bit architecture> int dc_mcf_ap_info_uoc(long flags, char *apname, struct DC_MCFAPINFO_UOC *apinfo)	int dc_mcf_ap_info_uoc(DCLONG flags, char *apname, struct DC_MCFAPINFO_UOC *apinfo)
<For 64-bit architecture> int dc_mcf_ap_info_uoc(int flags, char *apname, struct DC_MCFAPINFO_UOC *apinfo)	

(b) K&R C

Version 6 or earlier	Version 7
<For 32-bit architecture> int dc_mcf_ap_info_uoc(flags,apname, apinfo) long flags; char *apname; struct DC_MCFAPINFO_UOC *apinfo;	int dc_mcf_ap_info_uoc(flags,apname, apinfo) DCLONG flags; char *apname; struct DC_MCFAPINFO_UOC *apinfo;

Version 6 or earlier	Version 7
<For 64-bit architecture> int dc_mcf_ap_info_uoc(flags,apname, apinfo) int flags; char *apname; struct DC_MCFAPINFO_UOC *apinfo;	

(c) Arguments whose value is returned from OpenTP1

- **apinfo**

Version 6 or earlier	Version 7

<For 32-bit architecture>
struct DC_MCFAPINFO_UOC {
char mcf_apinfo[4];
long mcf_resv00;
char mcf_ap_name[9];
char mcf_ap_mcfid[3];
char mcf_resv01[4];
long mcf_ap_stat;
long mcf_ap_type;
long mcf_ap_msgcnt;
char mcf_sg_name[32];
long mcf_sg_stat;
long mcf_sg_hold;
long mcf_sg_msgcnt;
char mcf_sv_name[32];
long mcf_sv_stat;
long mcf_ap_ntmetim;
long mcf_ap_tempsize;
long mcf_ap_max_msgcnt;
long mcf_ap_trnmode;
long mcf_ap_quekind;
char mcf_resv02[64];
};
	struct DC_MCFAPINFO_UOC {
	char mcf_apinfo[4];
	DCLONG mcf_resv00;
	char mcf_ap_name[9];
	char mcf_ap_mcfid[3];
	char mcf_resv01[4];
	DCLONG mcf_ap_stat;
	DCLONG mcf_ap_type;
	DCLONG mcf_ap_msgcnt;
	char mcf_sg_name[32];
	DCLONG mcf_sg_stat;
	DCLONG mcf_sg_hold;
	DCLONG mcf_sg_msgcnt;
	char mcf_sv_name[32];
	DCLONG mcf_sv_stat;
	DCLONG mcf_ap_ntmetim;
	DCLONG mcf_ap_tempsize;
	DCLONG mcf_ap_max_msgcnt;
	DCLONG mcf_ap_trnmode;
	DCLONG mcf_ap_quekind;
	char mcf_resv02[64];
	};

Version 6 or earlier	Version 7

<For 64-bit architecture>
struct DC_MCFAPINFO_UOC {
char mcf_apinfo[4];
int mcf_resv00;
char mcf_ap_name[9];
char mcf_ap_mcfid[3];
char mcf_resv01[4];
int mcf_ap_stat;
int mcf_ap_type;
int mcf_ap_msgcnt;
char mcf_sg_name[32];
int mcf_sg_stat;
int mcf_sg_hold;
int mcf_sg_msgcnt;
char mcf_sv_name[32];
int mcf_sv_stat;
int mcf_ap_ntmetim;
int mcf_ap_tempsize;
int mcf_ap_max_msgcnt;
int mcf_ap_trnmode;
int mcf_ap_quekind;
char mcf_resv02[64];
};	

(3) dc_mcf_close - Close the MCF environment

(a) ANSI C, C++

Version 6 or earlier	Version 7
<For 32-bit architecture> void dc_mcf_close(long flags)	void dc_mcf_close(DCLONG flags)
<For 64-bit architecture> void dc_mcf_close(int flags)	

(b) K&R C

Version 6 or earlier	Version 7
<For 32-bit architecture> void dc_mcf_close(flags) long flags;	void dc_mcf_close(flags) DCLONG flags;
<For 64-bit architecture> void dc_mcf_close(flags) int flags;	

(4) dc_mcf_commit - Commit an MHP

(a) ANSI C, C++

Version 6 or earlier	Version 7
<For 32-bit architecture> `int dc_mcf_commit(long action)`	`int dc_mcf_commit(DCLONG action)`
<For 64-bit architecture> `int dc_mcf_commit(int action)`	

(b) K&R C

Version 6 or earlier	Version 7

<For 32-bit architecture>
`int dc_mcf_commit(action)`
`long action;`
	`int dc_mcf_commit(action)`
	`DCLONG action;`

<For 64-bit architecture>
`int dc_mcf_commit(action)`
`int action;`	

(5) dc_mcf_contend - Terminate continuous-inquiry response processing

(a) ANSI C, C++

Version 6 or earlier	Version 7

<For 32-bit architecture>
`int dc_mcf_contend(long action,`
`char *resv01)`
	`int dc_mcf_contend(DCLONG action,`
	`char *resv01)`

<For 64-bit architecture>
`int dc_mcf_contend(int action,`
`char *resv01)`	

(b) K&R C

Version 6 or earlier	Version 7

<For 32-bit architecture>
`int dc_mcf_contend(action,resv01)`
`long action;`
`char *resv01;`
	`int dc_mcf_contend(action,resv01)`
	`DCLONG action;`
	`char *resv01;`

<For 64-bit architecture>
`int dc_mcf_contend(action,resv01)`
`int action;`
`char *resv01;`	

(6) dc_mcf_execap - Activate an application program

(a) ANSI C, C++

Version 6 or earlier	Version 7

<For 32-bit architecture>
`int dc_mcf_execap(long action,`
`long commform,`
`char *resv01,`
`long active,`
`char *apnam,`
`char *comdata,`
`long cdataleng)`
	`int dc_mcf_execap(DCLONG action,`
	`DCLONG commform,`
	`char *resv01,`
	`DCLONG active,`
	`char *apnam,`
	`char *comdata,`
	`DCLONG cdataleng)`

<For 64-bit architecture>
`int dc_mcf_execap(int action,`
`int commform,`
`char *resv01,`
`int active,`
`char *apnam,`
`char *comdata,`
`int cdataleng)`	

(b) K&R C

Version 6 or earlier	Version 7

<For 32-bit architecture>
`int dc_mcf_execap(action,commform,`
`resv01,active,apnam,`
`comdata,cdataleng)`
`long action;`
`long commform;`
`char *resv01;`
`long active;`
`char *apnam;`
`char *comdata;`
`long cdataleng;`
	`int dc_mcf_execap(action,commform,`
	`resv01,active,apnam,`
	`comdata,cdataleng)`
	`DCLONG action;`
	`DCLONG commform;`
	`char *resv01;`
	`DCLONG active;`
	`char *apnam;`
	`char *comdata;`
	`DCLONG cdataleng;`

<For 64-bit architecture>
`int dc_mcf_execap(action,commform,`
`resv01,active,apnam,`
`comdata,cdataleng)`
`int action;`
`int commform;`
`char *resv01;`
`int active;`
`char *apnam;`
`char *comdata;`
`int cdataleng;`	

(7) dc_mcf_mainloop - Start an MHP service

(a) ANSI C, C++

Version 6 or earlier	Version 7
<For 32-bit architecture> `int dc_mcf_mainloop(long flags)`	`int dc_mcf_mainloop(DCLONG flags)`
<For 64-bit architecture> `int dc_mcf_mainloop(int flags)`	

(b) K&R C

Version 6 or earlier	Version 7

<For 32-bit architecture>
`int dc_mcf_mainloop(flags)`
`long flags;`
	`int dc_mcf_mainloop(flags)`
	`DCLONG flags;`

<For 64-bit architecture>
`int dc_mcf_mainloop(flags)`
`int flags;`	

(8) dc_mcf_open - Open the MCF environment

(a) ANSI C, C++

Version 6 or earlier	Version 7
<For 32-bit architecture> `int dc_mcf_open(long flags)`	`int dc_mcf_open(DCLONG flags)`
<For 64-bit architecture> `int dc_mcf_open(int flags)`	

(b) K&R C

Version 6 or earlier	Version 7

<For 32-bit architecture>
`int dc_mcf_open(flags)`
`long flags;`
	`int dc_mcf_open(flags)`
	`DCLONG flags;`

<For 64-bit architecture>
`int dc_mcf_open(flags)`
`int flags;`	

(9) dc_mcf_receive - Receive a message

(a) ANSI C, C++

Version 6 or earlier	Version 7

<For 32-bit architecture>
`int dc_mcf_receive(long action,`
`long commform,`
`char *termnam,`
`char *resv01,`
`char *recvdata,`
`long *rdataleng,`
`long inbufleng,`
`long *time)`
	`int dc_mcf_receive(DCLONG action,`
	`DCLONG commform,`
	`char *termnam,`
	`char *resv01,`
	`char *recvdata,`
	`DCLONG *rdataleng,`
	`DCLONG inbufleng,`
	`DCLONG *time)`

<For 64-bit architecture>
`int dc_mcf_receive(int action,`
`int commform,`
`char *termnam,`
`char *resv01,`
`char *recvdata,`
`int *rdataleng,`
`int inbufleng,`
`int *time)`	

(b) K&R C

Version 6 or earlier	Version 7

<For 32-bit architecture>
`int dc_mcf_receive(action,commform,`
`termnam,resv01,`
`recvdata,rdataleng,`
`inbufleng,time)`
`long action;`
`long commform;`
`char *termnam;`
`char *resv01;`
`char *recvdata;`
`long *rdataleng;`
`long inbufleng;`
`long *time;`
	`int dc_mcf_receive(action,commform,`
	`termnam,resv01,`
	`recvdata,rdataleng,`
	`inbufleng,time)`
	`DCLONG action;`
	`DCLONG commform;`
	`char *termnam;`
	`char *resv01;`
	`char *recvdata;`
	`DCLONG *rdataleng;`
	`DCLONG inbufleng;`
	`DCLONG *time;`

Version 6 or earlier	Version 7
<For 64-bit architecture> int dc_mcf_receive(action,commform, termnam,resv01, recvdata,rdataleng, inbufleng,time) int action; int commform; char *termnam; char *resv01; char *recvdata; int *rdataleng; int inbufleng; int *time;	

(10) dc_mcf_rollback - Enable MHP rollback

(a) ANSI C, C++

Version 6 or earlier	Version 7
<For 32-bit architecture> int dc_mcf_rollback(long action)	int dc_mcf_rollback(DCLONG action)
<For 64-bit architecture> int dc_mcf_rollback(int action)	

(b) K&R C

Version 6 or earlier	Version 7
<For 32-bit architecture> int dc_mcf_rollback(action) long action;	int dc_mcf_rollback(action) DCLONG action;
<For 64-bit architecture> int dc_mcf_rollback(action) int action;	

(11) dc_mcf_tempget - Accept temporarily-stored data

(a) ANSI C, C++

Version 6 or earlier	Version 7
<For 32-bit architecture> int dc_mcf_tempget(long action, char *getdata, long gtempleng, long *gdataleng, char *resv01)	int dc_mcf_tempget(DCLONG action, char *getdata, DCLONG gtempleng, DCLONG *gdataleng, char *resv01)

Version 6 or earlier	Version 7

<For 64-bit architecture>
`int dc_mcf_tempget(int action,`
`char *getdata,`
`int gtempleng,`
`int *gdataleng,`
`char *resv01)`	

(b) K&R C

Version 6 or earlier	Version 7

<For 32-bit architecture>
`int dc_mcf_tempget(action,getdata,`
`gtempleng,gdataleng,`
`resv01)`
`long action;`
`char *getdata;`
`long gtempleng;`
`long *gdataleng;`
`char *resv01;`
	`int dc_mcf_tempget(action,getdata,`
	`gtempleng,gdataleng,`
	`resv01)`
	`DCLONG action;`
	`char *getdata;`
	`DCLONG gtempleng;`
	`DCLONG *gdataleng;`
	`char *resv01;`

<For 64-bit architecture>
`int dc_mcf_tempget(action,getdata,`
`gtempleng,gdataleng,`
`resv01)`
`int action;`
`char *getdata;`
`int gtempleng;`
`int *gdataleng;`
`char *resv01;`	

(12) dc_mcf_tempput - Update temporarily-stored data

(a) ANSI C, C++

Version 6 or earlier	Version 7

<For 32-bit architecture>
`int dc_mcf_tempput(long action,`
`char *putdata,`
`long pdataleng,`
`char *resv01)`
	`int dc_mcf_tempput(DCLONG action,`
	`char *putdata,`
	`DCLONG pdataleng,`
	`char *resv01)`

<For 64-bit architecture>
`int dc_mcf_tempput(int action,`
`char *putdata,`
`int pdataleng,`
`char *resv01)`	

(b) K&R C

Version 6 or earlier	Version 7

<For 32-bit architecture>
`int dc_mcf_tempput(action,putdata,`
`pdataleng,resv01)`
`long action;`
`char *putdata;`
`long pdataleng;`
`char *resv01;`
	`int dc_mcf_tempput(action,putdata,`
	`pdataleng,resv01)`
	`DCLONG action;`
	`char *putdata;`
	`DCLONG pdataleng;`
	`char *resv01;`

<For 64-bit architecture>
`int dc_mcf_tempput(action,putdata,`
`pdataleng,resv01)`
`int action;`
`char *putdata;`
`int pdataleng;`
`char *resv01;`	

(13) dc_mcf_timer_cancel - Cancel user timer monitoring

(a) ANSI C, C++

Version 6 or earlier	Version 7

<For 32-bit architecture>
`int dc_mcf_timer_cancel(long flags,`
`long timer_id,`
`char *lename)`
	`int dc_mcf_timer_cancel(DCLONG flags,`
	`DCLONG timer_id,`
	`char *lename)`

<For 64-bit architecture>
`int dc_mcf_timer_cancel(int flags,`
`int timer_id,`
`char *lename)`	

(b) K&R C

Version 6 or earlier	Version 7

<For 32-bit architecture>
`int dc_mcf_timer_cancel(flags,`
`timer_id,`
`lename)`
`long flags;`
`long timer_id;`
`char *lename;`
	`int dc_mcf_timer_cancel(flags,`
	`timer_id,`
	`lename)`
	`DCLONG flags;`
	`DCLONG timer_id;`
	`char *lename;`

<For 64-bit architecture>
`int dc_mcf_timer_cancel(flags,`
`timer_id,`
`lename)`
`int flags;`
`int timer_id;`
`char *lename;`	

(14) dc_mcf_timer_set - Set user timer monitoring

(a) ANSI C, C++

Version 6 or earlier	Version 7

<For 32-bit architecture>
`int dc_mcf_timer_set(long flags,`
`long timer,`
`long timer_id,`
`char *lename,`
`char *apname,`
`char *data,`
`long data_leng,`
`long resv01,`
`long resv02)`
	`int dc_mcf_timer_set(DCLONG flags,`
	`DCLONG timer,`
	`DCLONG timer_id,`
	`char *lename,`
	`char *apname,`
	`char *data,`
	`DCLONG data_leng,`
	`DCLONG resv01,`
	`DCLONG resv02)`

<For 64-bit architecture>
`int dc_mcf_timer_set(int flags,`
`int timer,`
`int timer_id,`
`char *lename,`
`char *apname,`
`char *data,`
`int data_leng,`
`int resv01,`
`int resv02)`	

(b) K&R C

Version 6 or earlier	Version 7

<For 32-bit architecture>
`int dc_mcf_timer_set(flags,timer,`
`timer_id,lename,`
`apname,data,data_leng,`
`resv01,resv02)`
`long flags;`
`long timer;`
`long timer_id;`
`char *lename;`
`char *apname;`
`char *data;`
`long data_leng;`
`long resv01;`
`long resv02;`
	`int dc_mcf_timer_set(flags,timer,`
	`timer_id,lename,apname,`
	`data,data_leng,resv01,`
	`resv02)`
	`DCLONG flags;`
	`DCLONG timer;`
	`DCLONG timer_id;`
	`char *lename;`
	`char *apname;`
	`char *data;`
	`DCLONG data_leng;`
	`DCLONG resv01;`
	`DCLONG resv02;`

Version 6 or earlier	Version 7

<For 64-bit architecture>
`int dc_mcf_timer_set(flags,timer,`
`timer_id,lename,`
`apname,data,data_leng,`
`resv01,resv02)`
`int flags;`
`int timer;`
`int timer_id;`
`char *lename;`
`char *apname;`
`char *data;`
`int data_leng;`
`int resv01;`
`int resv02;`	

B.2 User exit routines

This section lists the changes to user exit routines.

(1) User exit routine that determines inheritance of timer-start messages

(a) Format

ANSI C, C++

Version 6 or earlier	Version 7
<For 32-bit architecture> `long uoc_func(dcmpsv_uoc_rtime *parm)`	`DCLONG uoc_func(dcmpsv_uoc_rtime *parm)`
<For 64-bit architecture> `int uoc_func(dcmpsv_uoc_rtime *parm)`	

K&R C

Version 6 or earlier	Version 7

<For 32-bit architecture>
`long uoc_func(parm)`
`dcmpsv_uoc_rtime *parm ;`
	`DCLONG uoc_func(parm)`
	`dcmpsv_uoc_rtime *parm ;`

<For 64-bit architecture>
`int uoc_func(parm)`
`dcmpsv_uoc_rtime *parm ;`	

(b) Parameters

Contents of `dcmpsv_uoc_rtime`

Version 6 or earlier	Version 7

<For 32-bit architecture>
`typedef struct {char le_name[9];`
`char reserve1[7];`
`char ap_name[9];`
`char reserve2[7];`
`long exec_time;`
`char ap_type;`
`char time_type;`
`char reserve3[26];`
`} dcmpsv_uoc_rtime;`
	`typedef struct {char le_name[9];`
	`char reserve1[7];`
	`char ap_name[9];`
	`char reserve2[7];`
	`DCLONG exec_time;`
	`char ap_type;`
	`char time_type;`
	`char reserve3[26];`
	`} dcmpsv_uoc_rtime;`

<For 64-bit architecture>
`typedef struct {char le_name[9];`
`char reserve1[7];`
`char ap_name[9];`
`char reserve2[7];`
`int exec_time;`
`char ap_type;`
`char time_type;`
`char reserve3[26];`
`} dcmpsv_uoc_rtime;`	

B.3 MCF event interfaces

This section lists the changes to the MCF event interfaces.

(1) Structure format of MCF event that reports discarding of a timer-start message (ERREVT4)

(a) Format of the common header for MCF event information

Version 6 or earlier	Version 7

<For 32-bit architecture>
`struct dc_mcf_evtheader {`
`char mcfevt_name[9] ;`
`char le_name[16] ;`
`char cn_name[9] ;`
`unsigned char format_kind;`
`char reserve01;`
`long time ;`
`};`
	`struct dc_mcf_evtheader {`
	`char mcfevt_name[9] ;`
	`char le_name[16] ;`
	`char cn_name[9] ;`
	`unsigned char format_kind;`
	`char reserve01;`
	`DCLONG time ;`
	`};`

Version 6 or earlier	Version 7

<For 64-bit architecture>
`struct dc_mcf_evtheader {`
`  char mcfevt_name[9] ;`
`  char le_name[16] ;`
`  char cn_name[9] ;`
`  unsigned char format_kind;`
`  char reserve01;`
`  int time ;`
`  };`	

B.4 Coding example for the MHP service function

This section lists the changes to the example of creating a user application program. The changes are indicated by underlines.

```
 10  /*
 20   * MHP service function
 30   */
 40  #include <stdio.h>
 50  #include <sys/types.h>
 60  #include <dcmcf.h>
 70  #include <dcrpc.h>
 80
 90  void svrA()
100  {
110     DCLONG action ;
120     DCLONG commform ;
130     DCLONG opcd ;
140     DCLONG active ;
150     char recvdata [1024] ;
160     DCLONG rdataleng ;
170     DCLONG time ;
180     DCLONG inbufleng ;
190     int  rtn_cod ;
200     DCLONG cdataleng ;
210     char termnam [10]  ;
220    static char execdata [32]  = "       SVRA EXECAP DATA" ;
230    static char senddata [32]  = "       SVRA REPLY DATA1" ;
240     static char resv01 [9]  = "\0" ;
250     static char resv02 [9]  = "\0" ;
260     static char resv03 [9]  = "\0" ;
270     static char apnam [9]  = "aprepB" ;
280
290     printf("*****   UAP START   *****\n") ;
300
310     printf("*****   MCF RECEIVE   *****\n") ;
320  /*
```

```
330   * MCF- RECEIVE (receive messages)
340   */
350       action = DCMCFFRST ;
360       commform = DCNOFLAGS ;
370       inbufleng = sizeof(recvdata) ;
380       rtn_cod = dc_mcf_receive(action, commform, termnam,
resv01,
390                 recvdata, &rdataleng, inbufleng, &time) ;
400       if(rtn_cod != DCMCFRTN_00000) {
410  /*
420   * MCF- ROLLBACK (error processing)
430   */
440     printf("dc_mcf_receive failed !! CODE = %d \n", rtn_cod)
;
450       rtn_cod = dc_mcf_rollback(DCMCFNRTN) ;
460       }
470
480       printf("*****   MCF EXECAP    *****\n") ;
490  /*
500   * MCF-EXECAP (start the application program)
510   */
520       action = DCMCFEMI | DCMCFJUST ;
530       commform = DCNOFLAGS ;
540       active = 0 ;
550       cdataleng = 16 ;
560       rtn_cod = dc_mcf_execap(action, commform, resv01,
active,
570                 apnam, comdata, cdataleng) ;
580       if(rtn_cod != DCMCFRTN_00000) {
590  /*
600   * MCF- ROLLBACK (error processing)
610   */
620      printf("dc_mcf_execap failed !! CODE = %d \n", rtn_cod)
;
630       rtn_cod = dc_mcf_rollback(DCMCFNRTN) ;
640       }
650
660       printf("*****   MCF REPLY   *****\n") ;
670  /*
680   * MCF-REPLY (send a response message)
690   */
700       action = DCMCFEMI ;
710       commform = DCNOFLAGS ;
720       opcd = DCNOFLAGS ;
730       cdataleng = 16 ;
740       rtn_cod = dc_mcf_reply(action, commform, resv01,
resv02,
750                 senddata, cdataleng, resv03, opcd) ;
```

```
760       if(rtn_cod != DCMCFRTN_00000) {
770  /*
780   * MCF- ROLLBACK (error processing)
790   */
800      printf("dc_mcf_reply failed !! CODE = %d \n", rtn_cod) ;
810       rtn_cod = dc_mcf_rollback(DCMCFNRTN) ;
820       }
830  }
```

Index

D

S

T

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)