

## **OpenTP1 Version 7**

### **Description**

3000-3-D50-30(E)

## ■ Relevant program product

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*.

This product was developed under a quality management system that has received ISO9001 and TickIT certification.

#### ■ Trademarks

AIX is a trademark of International Business Machines Corporation in the United States, other countries, or both.

AIX 5L is a trademark of International Business Machines Corporation in the United States, other countries, or both.

AMD, AMD Opteron, and combinations thereof, are trademarks of Advanced Micro Devices, Inc.

COBOL/2 is a trademark of International Business Machines Corporation in the United States, other countries, or both.

Gauntlet is a registered trademark of Network Associates, Inc. and/or its affiliates in the US and/or other countries.

HP-UX is a product name of Hewlett-Packard Company.

Itanium is a trademark of Intel Corporation in the United States and other countries.

Java is either a registered trademark or a trademark of Oracle and/or its affiliates.

Linux(R) is the registered trademark of Linus Torvalds in the U.S. and other countries.

Microsoft is either a registered trademark or a trademark of Microsoft Corporation in the United States and/or other countries.

MQSeries is a trademark of International Business Machines Corporation in the United States, other countries, or both.

MS-DOS is a registered trademark of Microsoft Corp. in the U.S. and other countries.

ORACLE is either a registered trademark or a trademark of Oracle and/or its affiliates.

Oracle is either a registered trademark or a trademark of Oracle Corporation and/or its affiliates.

Oracle and Oracle 10g are either registered trademarks or trademarks of Oracle and/or its affiliates.

Oracle and Oracle9i are either registered trademarks or trademarks of Oracle and/or its affiliates.

OSF is a trademark of the Open Software Foundation, Inc.

Red Hat is a trademark or a registered trademark of Red Hat Inc. in the United States and other countries.

Solaris is either a registered trademark or a trademark of Oracle and/or its affiliates.

UNIX is a registered trademark of The Open Group in the United States and other countries.

WebSphere is a trademark of International Business Machines Corporation in the United States, other countries, or both.

Windows is either a registered trademark or a trademark of Microsoft Corporation in the United States and/or other countries.

Windows Server is either a registered trademark or a trademark of Microsoft Corporation in the United States and/or other countries.

Windows Vista is either a registered trademark or a trademark of Microsoft Corporation in the United States and/or other countries.

X/Open is a registered trademark of The Open Group in the U.K. and other countries.

Other product and company names mentioned in this document may be the trademarks of their respective owners. Throughout this document Hitachi has attempted to distinguish trademarks from descriptive terms by writing the name with the capitalization used by the manufacturer, or by writing the name with initial capital letters. Hitachi cannot attest to the accuracy of this information. Use of a trademark in this document should not be regarded as affecting the validity of the trademark.

Portions of this document are extracted from *X/Open CAE Specification System Interfaces and Headers, Issue 4*, (C202 ISBN 1-872630-47-2) Copyright (C) July 1992, X/Open Company Limited with the permission of X/Open; part of which 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.

Portions of this document are extracted from *X/Open Preliminary Specification Distributed Transaction Processing: The TxRPC Specification* (P305 ISBN 1 85912 000 8) Copyright (C) July 1993, X/Open Company Limited with the permission of X/Open.

This documentation 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.

Other product and company names mentioned in this document may be the trademarks of their respective owners. Throughout this document Hitachi has attempted to distinguish trademarks from descriptive terms by writing the name with the capitalization used by the manufacturer, or by writing the name with initial capital letters. Hitachi cannot attest to the accuracy of this information. Use of a trademark in this document should not be regarded as affecting the validity of the trademark.

#### ■ Restrictions

Information in this document is subject to change without notice and does not represent a commitment on the part of Hitachi. The software described in this manual is furnished according to a license agreement with Hitachi. The license agreement contains all of the terms and conditions governing your use of the software and documentation, including all warranty rights, limitations of liability, and disclaimers of warranty.

Material contained in this document may describe Hitachi products not available or features not available in your country.

No part of this material may be reproduced in any form or by any means without permission in writing from the publisher.

Printed in Japan.

#### ■ Edition history

Edition 1 (3000-3-D50(E)): June 2006

Edition 3 (3000-3-D50-30(E)): October 2010

#### ■ Copyright

All Rights Reserved. Copyright (C) 2006, 2010, Hitachi, Ltd.

## Summary of amendments

The following table lists changes in this manual (3000-3-D50-30(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
A performance verification trace (JNL performance verification trace) can now be output by using the journal service.	1.4.3, 4.1.1(2), 5.3.6 and 5.3.6(6)
A performance verification trace (LCK performance verification trace) for locking events that use lock services can now be output.	1.4.3, 4.1.1(2), 5.3.6 and 5.3.6(7)
An explanation of global domain has been added.	3.2.1(5)(b)
A function for prioritizing using service information of a specific node (service information prioritizing function) has been added.	3.2.2, 3.2.2(2), and Appendix E
An explanation of the causes for the MCF event that reports UAP abnormal termination and the MCF event that reports discarding of an unprocessed message has been changed.	3.3.7(3)(a)
An explanation of fall-back operation has been changed.	3.3.8(2)
An explanation regarding cases in which messages cannot be recovered has been added.	3.3.9(4)(b)
An explanation of the send order of messages has been changed.	3.3.10
An explanation of the function for receiving data that is reported by the dc_rpc_cltsend function has been changed.	3.5.1(2)
An explanation of the differences between an OpenTP1 file system and an OS file system has been changed.	4.1.1(1) Table 4-2
An event trace (FIL event trace) can now be output by requesting access to OpenTP1 files.	4.1.1(2), 5.3.6, and 5.3.6(13)
An explanation of the shared memory size for the DAM service when using a DAM file that has been specified for cacheless access has been changed.	4.4.1(15)
The number of services for acquisition of performance verification has been changed. Related definitions have also been changed.	7.1
A description of the system services that can specify port numbers for receiving data has been added.	7.3.1
The statuses of application timer start requests can now be displayed.	Appendix B Table B-2
The prctee process, which redirects OpenTP1 standard output and standard error output, can now be stopped and restarted.	Appendix B Table B-2



<b>Changes</b>	<b>Location</b>
The statuses of user timer monitoring can now be displayed.	<i>Appendix B Table B-2</i>
Changes in the following product versions are listed: <ul style="list-style-type: none"> <li>• TP1/Server Base 07-03</li> <li>• TP1/Message Control 07-03 and TP1/NET/Library 07-04</li> <li>• TP1/Message Control 07-02 and TP1/NET/Library 07-03</li> <li>• TP1/Message Control 07-01 and TP1/NET/Library 07-01</li> </ul>	<i>Appendix C.1, Appendix C.2, and Appendix C.3</i>
An overview regarding the processing of remote procedure calls has been added.	<i>Appendix D</i>

The following table lists changes in this manual (3000-3-D50-30(E)) and product changes related to this manual for uCosminexus TP1/Message Control 07-02, and uCosminexus TP1/NET/Library 07-03.

<b>Changes</b>	<b>Location</b>
A function for dynamic loading of service functions can now be used by using MHP.	<i>2.6.2(3)(b), 3.8, 3.8.1 and Appendix E</i>
The following operations can now be performed by using library functions: <ul style="list-style-type: none"> <li>• Displaying the status of, establishing, and releasing connections</li> <li>• Displaying the status of requests to establish server connections, and starting or terminating reception of those requests</li> <li>• Deletion of application timer start requests</li> <li>• Displaying the status of a logical terminal, shutting it down, and releasing it from shutdown, and deleting the output queue</li> <li>• Displaying the status of the MCF communication service or the application start service</li> </ul>	<i>3.3.7(3)(a), 3.3.9(3) and Appendix B Table B-1</i>
A message is now output when memory is automatically added from the unused area if MCF static shared memory is insufficient.	<i>7.2.2(4)</i>
The status of a network that is related to a message exchange with a remote system can now be displayed.	<i>Appendix B Table B-2</i>

The following table lists changes in this manual (3000-3-D50-30(E)) and product changes related to this manual for uCosminexus TP1/Message Control 07-01, and uCosminexus TP1/NET/Library 07-01.

<b>Changes</b>	<b>Location</b>
A performance verification trace (MCF performance verification trace) can now be output by main events during message exchanges.	<i>1.4.3, 4.1.1(2), 5.3.6, and 5.3.6(8)</i>
Reception of requests to establish server connections can now be manually started and terminated.	<i>Appendix B Table B-2</i>

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

The following table lists changes in the manual (3000-3-D50-20(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.

<b>Changes</b>
A performance verification trace (PRF trace) can now be output by the XA resource service.
A function that enables service functions to be loaded dynamically has been added.
Items that can be monitored by the timer monitoring facility have been added. OpenTP1 definitions related to the monitored items have been changed accordingly.
The schedule service can now be implemented on a service-by-service basis.
The description of usage of the User Authentication facility in TP1/Client/P for MS-DOS has been deleted.
The description of the remote API facility has been changed.
An audit log output function has been added.
A function that allows parallel access to system journal files has been added.
An event trace can now be output by the name service.
An event trace can now be output by a process service
A function that allows the system to operate without using system journal files (non-journal operation function) has been added. With this addition, the valid number of system service processes has been changed.
Changes in the following product versions are described: <ul style="list-style-type: none"> <li>• TP1/Server Base 07-02</li> <li>• TP1/Message Control 07-01 and TP1/NET/Library 07-01</li> </ul>

The following table lists changes in the manual (3000-3-D50-20(E)) and product changes related to that manual for uCosminexus TP1/Server Base 07-01.

<b>Changes</b>
A function (MSDTC linkage) that enables transactional linkage by means of two-phase commit between OpenTP1 and an application that runs on .NET Framework has been added.
A function has been added for displaying the product name, version number, and other information about products operating in environments set up in the OpenTP1 directory. With this addition, the <code>dcpplist</code> command has been added.
Changes to functions, definitions, commands, and defaults when upgrading to a later version are described.

---

# Preface

---

This manual provides a general overview of the OpenTP1 software product.

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.

TP1/Message Queue is implemented based on the specifications of the MQI, MQFAP, and MQ clusters in WebSphere MQ (formerly *MQSeries*) under license from International Business Machines Corporation.

## Intended readers

This manual is intended as a general overview that can be used by the following types of readers:

- General readers who want a non-technical overview of OpenTP1. These persons should read Chapter 1. Introduction. This chapter assumes a general understanding of computers and computing systems.
- OpenTP1 administrators who set up and manage OpenTP1 systems.
- Program developers interested in developing programs for OpenTP1 systems.

## Organization of this manual

This manual is organized into the following chapters and appendixes:

### *1. Introduction*

This chapter provides a general overview of OpenTP1 and online transaction processing.

### *2. Application Processing Modes*

This chapter describes the modes of application processing that can be performed in OpenTP1, related products, and the types of user application programs used in an OpenTP1 system.

### *3. Functions*

This chapter describes the service functions provided by OpenTP1.

### *4. File System*

This chapter describes the OpenTP1 file system, files (such as OpenTP1 files and user files), and the IST service.

### *5. Overview of Setup, Use, and Error Recovery*

This chapter describes the actions taken by OpenTP1 or users when setting up or using OpenTP1, or when recovering from some failure or error.

### *6. Using Multiple Instances of OpenTP1*

This chapter describes the OpenTP1 features that can be used when constructing systems that require multiple instances of OpenTP1.

### *7. System Resources*

This appendix describes OpenTP1 process structure and memory structure.

#### *A. Communication Protocol Products for Use with TP1/Message Control*

This appendix describes the OpenTP1 program products that support specific communication protocols, required for communication based on message exchange.

#### *B. Library Functions and Commands*

This appendix lists and describes the library functions and commands available in OpenTP1.

#### *C. Version Changes*

This appendix describes the changes in definitions, functions, and commands between OpenTP1 versions.

#### *D. Overview of Remote Procedure Call Processing*

This appendix provides an overview of remote procedure call processing.

#### *E. Glossary*

The glossary defines terms and phrases used in this manual.

## **Related publications**

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

### **OpenTP1 products**

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

(3000-3-D55(E))

- *OpenTP1 Version 7 Messages* (3000-3-D56(E))
- *OpenTP1 Version 7 Tester and UAP Trace User's Guide* (3000-3-D57(E))
- *OpenTP1 Version 7 TP1/Client User's Guide TP1/Client/W, TP1/Client/P* (3000-3-D58(E))
- *OpenTP1 Version 7 TP1/Client User's Guide TP1/Client/J* (3000-3-D59(E))
- *OpenTP1 Version 7 TP1/LiNK User's Guide* (3000-3-D60(E))<sup>#</sup>
- *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))<sup>#</sup>
- *OpenTP1 Version 7 TP1/Message Queue Messages* (3000-3-D91(E))<sup>#</sup>
- *OpenTP1 Version 7 TP1/Message Queue Application Programming Guide* (3000-3-D92(E))<sup>#</sup>
- *OpenTP1 Version 7 TP1/Message Queue Application Programming Reference* (3000-3-D93(E))<sup>#</sup>

#### **Other OpenTP1 products**

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

#### **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.

#

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)	

Abbreviation		Full name or meaning
		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)
		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 <sup>(R)</sup> MS-DOS <sup>(R)</sup>
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)

<b>Abbreviation</b>	<b>Full name or meaning</b>
TP1/Extension 1	uCosminexus TP1/Extension 1
	uCosminexus TP1/Extension 1(64)
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



Abbreviation		Full name or meaning
TP1/NET/SLU - TypeP2	TP1/NET/ Secondary Logical Unit - TypeP2	uCosminexus TP1/NET/Secondary Logical Unit - TypeP2
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 <sup>(R)</sup> Windows <sup>(R)</sup> 2000 Advanced Server Operating System
		Microsoft <sup>(R)</sup> Windows <sup>(R)</sup> 2000 Datacenter Server Operating System
		Microsoft <sup>(R)</sup> Windows <sup>(R)</sup> 2000 Professional Operating System
		Microsoft <sup>(R)</sup> Windows <sup>(R)</sup> 2000 Server Operating System
Windows Server 2003		Microsoft <sup>(R)</sup> Windows Server <sup>(R)</sup> 2003, Datacenter Edition
		Microsoft <sup>(R)</sup> Windows Server <sup>(R)</sup> 2003, Enterprise Edition
		Microsoft <sup>(R)</sup> Windows Server <sup>(R)</sup> 2003, Standard Edition
Windows Server 2003 R2		Microsoft <sup>(R)</sup> Windows Server <sup>(R)</sup> 2003 R2, Enterprise Edition

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

- If there is no difference in OS functionality, the term *Windows* is used to indicate Windows 2000, Windows Server 2003, Windows Server 2008, Windows XP, and Windows Vista.
- The term UNIX is used to indicate AIX, HP-UX, Linux, and Solaris.

## Conventions: Acronyms

This manual also uses the following acronyms:

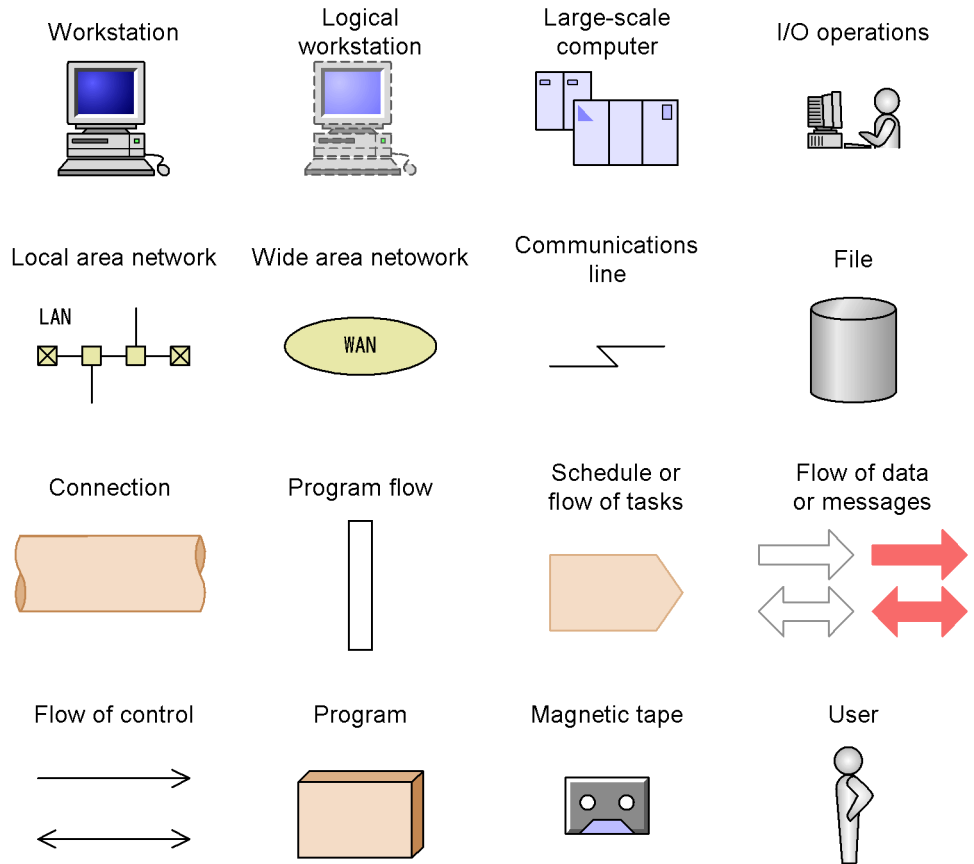
Acronym	Full name or meaning
ANSI	American National Standards Institute
AP	Application Program
API	Application Programming Interface
C/S	Client/Server
CGI	Common Gateway Interface
CPU	Central Processing Unit
CRM	Communication Resource Manager
CUP	Client User Program
DAM	Direct Access Method
DB	Database
DBMS	Database Management System
DCE	Distributed Computing Environment
DHCP	Dynamic Host Configuration Protocol
DID	Distributed Identifier
DML	Data Manipulation Language
DNS	Domain Name System
DPM	ServerConductor/DeploymentManager
DPOS	Distributed Data Processing Operating System
DTP	Distributed Transaction Processing
EX	Exclusive
FDDI	Fiber Distributed Data Interface
FEP	Front End Processor
FIFO	First-In-First-Out
FRC	File Recovery
GUI	Graphical User Interface

<b>Acronym</b>	<b>Full name or meaning</b>
HA	High Availability
HDLC-NRM	HDLC-Normal Response Mode
HTML	Hyper Text Markup Language
I/O	Input/Output
ID	Identifier
IDL	Interface Definition Language
IP	Internet Protocol
ISAM	Indexed Sequential Access Method
IST	Internode Shared Table
J2EE	Java 2 Enterprise Edition
JCA	J2EE Connector Architecture
JDBC	Java DataBase Connectivity
LAN	Local Area Network
MCF	Message Control Facility
MHP	Message Handling Program
MIA	Multivendor Integration Architecture
MQA	Message Queue Access
MQI	Message Queue Interface
MSDTC	Microsoft Distributed Transaction Coordinator
NIF/HNA	Network Interface Feature/Hitachi Network Architecture
NIF/OSI	Network Interface Feature/OSI
NIS	Network Information Service
OLTP	Online Transaction Processing
OS	Operating System
OSI	Open Systems Interconnection
OSI TP	Open Systems Interconnection Transaction Processing
PC	Personal Computer

<b>Acronym</b>	<b>Full name or meaning</b>
PR	Protected Retrieve
PRF	Performance
PVC	Permanent Virtual Circuit
RI	Recovery Information
RM	Resource Manager
RPC	Remote Procedure Call
RTS	Real Time Statistic
SCSI	Small Computer Systems Interface
SNA	Systems Network Architecture
SPP	Service Providing Program
SRF	Server Recovery Journal File
STDL	Structured Transaction Definition Language
SUP	Service Using Program
TAM	Table Access Method
TCO	Total Cost of Ownership
TCP/IP	Transmission Control Protocol/Internet Protocol
TP	Transaction Processing
TRF	Transaction Recovery journal File
UAP	User Application Program
UOC	User Own Coding
VM	Virtual Machine
WAN	Wide Area Network
WS	Workstation
WWW	World Wide Web
XA	Extended Architecture
XAR	Extended Architecture Resource

## Conventions: Diagrams

This manual uses the following conventions in diagrams:



## Conventions: Differences in installation directory paths

This manual uses the notation `/BeTRAN` to indicate the OpenTP1 installation directory. The actual installation directory differs depending on the operating system. Use the following table to determine the actual installation directory for your OS.

As written in this manual	Actual directory for each OS		
	AIX, HP-UX, and Solaris	Linux	Windows
<code>/BeTRAN</code>	<code>/BeTRAN</code>	<code>/opt/OpenTP1</code>	The directory in which OpenTP1 was installed

## Conventions: Fonts and symbols

The following table explains the fonts used in this manual:

Font	Convention
<b>Bold</b>	<p><b>Bold</b> 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:</p> <ul style="list-style-type: none"> <li>• From the <b>File</b> menu, choose <b>Open</b>.</li> <li>• Click the <b>Cancel</b> button.</li> <li>• In the <b>Enter name</b> entry box, type your name.</li> </ul>
<i>Italics</i>	<p><i>Italics</i> are used to indicate a placeholder for some actual text to be provided by the user or system. For example:</p> <ul style="list-style-type: none"> <li>• Write the command as follows: <code>copy source-file target-file</code></li> <li>• The following message appears: A file was not found. (file = <i>file-name</i>)</li> </ul> <p><i>Italics</i> are also used for emphasis. For example:</p> <ul style="list-style-type: none"> <li>• Do <i>not</i> delete the configuration file.</li> </ul>
Code font	<p>A code font indicates text that the user enters without change, or text (such as messages) output by the system. For example:</p> <ul style="list-style-type: none"> <li>• At the prompt, enter <code>dir</code>.</li> <li>• Use the <code>send</code> command to send mail.</li> <li>• The following message is displayed: <code>The password is incorrect.</code></li> </ul>

The following table explains the symbols used in this manual:

Symbol	Convention
	<p>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.</p>
{ }	<p>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.</p>
[ ]	<p>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.</p>
...	<p>In coding, an ellipsis (...) indicates that one or more lines of coding are not shown for purposes of brevity.</p> <p>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.</p>

## 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<sup>2</sup> bytes.
- 1 GB (gigabyte) is 1,024<sup>3</sup> bytes.
- 1 TB (terabyte) is 1,024<sup>4</sup> 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	<i>\$aaaaaa</i> Example: \$DCDIR	<i>%aaaaaa%</i> 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*.



---

# Contents

---

<b>Preface</b>	<b>i</b>
Intended readers .....	i
Organization of this manual .....	i
Related publications .....	ii
Conventions: Abbreviations for product names .....	iv
Conventions: Acronyms .....	ix
Conventions: Diagrams .....	xii
Conventions: Differences in installation directory paths .....	xii
Conventions: Fonts and symbols.....	xiii
Conventions: KB, MB, GB, and TB .....	xiv
Conventions: Platform-specific notational differences .....	xiv
Conventions: Version numbers.....	xiv
<b>1. Introduction</b>	<b>1</b>
1.1 Overview of OpenTP1 .....	2
1.1.1 Distributed computing environment for transaction processing .....	3
1.1.2 Flexible system configuration .....	4
1.1.3 Achieving a large-scale system linked to a backbone system.....	5
1.1.4 Support for moving to open systems.....	6
1.1.5 Unrestricted .....	7
1.2 Examples of configurations possible with OpenTP1 systems.....	8
1.2.1 OpenTP1 in a LAN that uses client/server processing.....	8
1.2.2 OpenTP1 in front-end processors.....	9
1.2.3 OpenTP1 connected to a non-OpenTP1 system.....	10
1.3 OpenTP1 software products .....	12
1.3.1 List of OpenTP1 software products .....	12
1.3.2 OpenTP1 and the X/Open DTP model.....	16
1.4 OpenTP1 system services.....	20
1.4.1 Types of OpenTP1 services.....	20
1.4.2 OpenTP1 system services.....	20
1.4.3 OpenTP1 system definitions .....	23
<b>2. Application Processing Modes</b>	<b>27</b>
2.1 Overview of OpenTP1 communications .....	28
2.2 Processing in a client/server configuration.....	29
2.2.1 Communication via remote procedure calls.....	29
2.2.2 Using OpenTP1 client software on workstations and PCs.....	32
2.3 Processing in an MCF message-exchange configuration .....	34

2.3.1	Overview of MCF message exchange .....	34
2.3.2	Networks that can use MCF message exchange.....	35
2.3.3	MCF message-exchange configuration using the Extended Presentation facility.....	35
2.4	Processing in an MQA message-queuing configuration.....	38
2.4.1	Features of MQA message queuing.....	38
2.4.2	Overview of communication using MQA message queuing.....	39
2.4.3	Notes on use of the MQA message queuing.....	40
2.5	Other Hitachi software products usable with OpenTP1 .....	41
2.5.1	Job Management Partner 1 Integrated System Operation Management Facilities .....	41
2.5.2	SEWB3 Software Engineering Workbench.....	43
2.6	User application programs in OpenTP1 systems .....	45
2.6.1	User application programs and types of processing .....	45
2.6.2	Overview of user application programs.....	45
2.6.3	Cooperation of user processes with SPPs and MHPs.....	55
2.6.4	UAP testing and debugging facilities .....	55
2.7	Processing in an Internet-based configuration.....	57

### **3. Functions** 59

---

3.1	Transaction Control.....	60
3.1.1	Distributed transactions .....	60
3.1.2	RPCs, transaction branches, and global transactions .....	62
3.1.3	Commit and rollback operations.....	62
3.1.4	Two-phase commit.....	63
3.1.5	Transactions and UAPs.....	67
3.1.6	Transaction control based on the TX interface.....	70
3.1.7	Transaction control based on the XA resource service .....	70
3.1.8	XA interface .....	79
3.2	Processing in an OpenTP1 client/server configuration.....	81
3.2.1	Communication via RPCs that use the OpenTP1 library .....	81
3.2.2	Optional function for service information searches .....	90
3.2.3	Node management in OpenTP1.....	105
3.2.4	Communication via RPCs that use the XATMI interface.....	113
3.2.5	Communication via RPCs that use the TxRPC interface .....	116
3.3	Message Control .....	120
3.3.1	Overview of sending and receiving messages using MCF.....	120
3.3.2	Message structure .....	121
3.3.3	Application program structure and application name.....	121
3.3.4	Synchronous and asynchronous communication functions, and messages .....	122
3.3.5	Messages independent of the above communication modes .....	124
3.3.6	Message-control transactions .....	125
3.3.7	Starting user application programs .....	126
3.3.8	MCF message queues and the sending and receiving of messages .....	134

3.3.9	Message exchange by user application programs .....	136
3.3.10	Order of sending MCF messages .....	143
3.3.11	Partially changing the MCF communication service during operation of OpenTP1 .....	145
3.3.12	MCF capabilities that are not supported in Windows .....	146
3.4	Scheduling .....	147
3.4.1	Scheduling requests to service-providing programs .....	147
3.4.2	Scheduling MCF messages to message-handling programs .....	156
3.4.3	Process control and the Multiserver facility .....	162
3.4.4	Saving shared memory in sharing a buffer area .....	177
3.4.5	Example of process control with the Multiserver facility .....	179
3.5	OpenTP1 client facility (TP1/Client) .....	182
3.5.1	Remote procedure calls of TP1/Client .....	183
3.5.2	MCF message exchange using the TCP/IP protocol .....	187
3.5.3	Communication with XDM/DCCM3 .....	188
3.6	Client/server communications using OSI TP .....	189
3.6.1	OpenTP1's remote system .....	189
3.6.2	Route used for communication .....	190
3.6.3	Application programs used for communication .....	191
3.6.4	Overview of environment setup .....	195
3.6.5	In the event of a failure .....	195
3.7	Remote API facility .....	196
3.7.1	Example of using the remote API facility .....	199
3.7.2	Permanent connection .....	201
3.7.3	Connection mode .....	202
3.7.4	RAP-processing client manager .....	203
3.7.5	Definitions necessary for using the remote API facility .....	203
3.7.6	Prerequisites for using the XA resource service .....	204
3.8	Dynamic loading of service functions .....	205
3.8.1	Examples of using dynamic loading of service functions .....	205
3.8.2	Preparation required for using dynamic loading of service functions .....	209
3.9	Additional Features .....	211
3.9.1	Locking resources .....	211
3.9.2	Acquisition of a user journal .....	215
3.9.3	Journal maintenance facilities .....	216
3.9.4	Obtaining the message log .....	217
3.9.5	Reporting a message log .....	219
3.9.6	Controlling resource managers not provided by OpenTP1 .....	220
3.9.7	Uptime statistics .....	222
3.9.8	Real-time statistics service .....	224
3.10	System operations using scenario templates .....	227
3.11	System monitoring using audit logs .....	229

<b>4. File System</b>	<b>233</b>
4.1 The OpenTP1 file system .....	234
4.1.1 Overview of the OpenTP1 file system .....	234
4.1.2 Creating an OpenTP1 file system .....	240
4.1.3 Backing up and restoring OpenTP1 file systems.....	243
4.1.4 Protecting OpenTP1 files.....	243
4.1.5 Assigning an OpenTP1 file system .....	244
4.2 System files.....	246
4.2.1 System files: status files .....	246
4.2.2 System files: system journal files .....	249
4.2.3 System files: checkpoint dump files.....	259
4.2.4 System files: transaction recovery journal file .....	266
4.2.5 System files: server recovery journal file .....	267
4.2.6 System files: archive journal files .....	267
4.3 Queue files .....	273
4.3.1 Queue files: MCF message queue file.....	273
4.3.2 Queue files: MQA message queue file.....	274
4.4 User data files .....	276
4.4.1 User files: DAM files (TP1/FS/Direct Access) .....	276
4.4.2 User files: TAM files (TP1/FS/Table Access) .....	293
4.4.3 IST service (TP1/Shared Table Access) .....	301
4.4.4 User files: ISAM files (ISAM and ISAM/B).....	306
4.4.5 Accessing database management systems .....	307
<b>5. Overview of Setup, Use, and Error Recovery</b>	<b>311</b>
5.1 Setting up an OpenTP1 system.....	312
5.1.1 Overview of environment settings.....	312
5.1.2 Environment setup tasks .....	314
5.2 Operating an OpenTP1 system .....	317
5.3 Failure and error recovery.....	321
5.3.1 Recovering from OpenTP1 system failures.....	321
5.3.2 Recovering from UAP failures .....	324
5.3.3 Recovering from file errors .....	326
5.3.4 Recovering from network errors.....	329
5.3.5 OpenTP1 monitoring and trace facilities.....	330
5.3.6 Analyzing the cause of an error.....	332
<b>6. Using Multiple Instances of OpenTP1</b>	<b>349</b>
6.1 The System Switchover facility .....	350
6.1.1 Overview of the System Switchover facility .....	350
6.1.2 OpenTP1 system configuration for using the System Switchover facility.	353
6.1.3 Procedure for system switching.....	354
6.1.4 Operating with the System Switchover facility .....	357

6.2	The Multinode facility .....	363
6.2.1	Overview of the Multinode facility .....	363
6.2.2	Available operations in the Multinode facility .....	366
6.2.3	Global Archive Journal facility .....	367
6.3	The MultiOpenTP1 .....	372
6.3.1	MultiOpenTP1 configuration .....	372
6.4	Multi-homed host configuration .....	374

## **7. System Resources** 377

---

7.1	OpenTP1 process structure .....	378
7.2	OpenTP1 memory structure .....	384
7.2.1	Local memory .....	384
7.2.2	Shared memory .....	384
7.3	TCP/IP resources that OpenTP1 uses .....	387
7.3.1	Port numbers used in OpenTP1 .....	387
7.3.2	How RPCs use ports .....	388
7.3.3	Calculating the number of ports .....	389
7.3.4	Restricting the number of ports .....	390
7.3.5	Temporary closing and user tasks .....	391
7.3.6	Monitoring a temporary closing request .....	392
7.3.7	Checking an execution status of temporary closing .....	392
7.3.8	Changes in the size of a resource when the number of sockets increases ...	394
7.3.9	Tuning the network environment .....	394
7.3.10	Cautions required when using DNS and NIS .....	395

## **Appendixes** 397

---

A.	Communication Protocol Products for Use with TP1/Message Control .....	398
A.1	OpenTP1 communication protocol products .....	398
A.2	Systems connected to protocol products .....	399
B.	Library Functions and Commands .....	401
C.	Version Changes .....	421
C.1	Changes in 07-03 .....	421
C.2	Changes in 07-02 .....	424
C.3	Changes in 07-01 .....	430
C.4	Changes in 07-00 .....	433
D.	Overview of Remote Procedure Call Processing .....	438
D.1	Overview of processing a remote procedure call to the local node .....	438
D.2	Overview of processing a remote procedure call to remote nodes .....	440
D.3	Overview of global search processing .....	444
D.4	Overview of service information registration and deletion processing .....	448
D.5	Overview of node-to-node forwarding processing .....	451
D.6	Overview of remote procedure call processing using the dcsvgdef definition command .....	453
E.	Glossary .....	456



---

## List of figures

---

Figure 1-1: Features of OpenTP1 .....	2
Figure 1-2: Conventional online transaction processing in a centralized system using a large-scale computer .....	3
Figure 1-3: Online transaction processing in a distributed environment using OpenTP1 .....	4
Figure 1-4: Flexible system configuration .....	5
Figure 1-5: Large-scale system incorporating a backbone system .....	6
Figure 1-6: OpenTP1 online transaction processing using the client/server model in a LAN ...	9
Figure 1-7: Example of a front-end processor (FEP) .....	10
Figure 1-8: OpenTP1 online transaction processing in a system using a large-scale host computer .....	11
Figure 1-9: OpenTP1 software products .....	12
Figure 1-10: The X/Open DTP model .....	17
Figure 1-11: Relationship between the X/Open DTP model and the OpenTP1 system .....	19
Figure 1-12: Organization of OpenTP1 system definitions .....	24
Figure 2-1: Overview of communication in a client/server configuration .....	30
Figure 2-2: Overview of communication based on TP1/Client .....	33
Figure 2-3: Overview of MCF message-exchange processing .....	35
Figure 2-4: An MCF message-exchange configuration that uses the Extended Presentation facility .....	36
Figure 2-5: Example of an OpenTP1 system configuration using MQA message queuing .....	38
Figure 2-6: Overview of MQA message queuing .....	39
Figure 2-7: OpenTP1 system configuration using Job Management Partner 1 .....	42
Figure 2-8: SUPs and SPPs in OpenTP1 systems .....	47
Figure 2-9: SPP structure (when using a stub) .....	49
Figure 2-10: SPP structure (when using dynamic loading of service functions) .....	50
Figure 2-11: MHP in an OpenTP1 system .....	51
Figure 2-12: MHP structure (when using a stub) .....	53
Figure 2-13: MHP structure (when performing dynamic loading of service functions) .....	54
Figure 2-14: Configuration with TP1/Web .....	57
Figure 3-1: A distributed transaction .....	61
Figure 3-2: A global transaction and transaction branches .....	62
Figure 3-3: Rollback operation on a transaction .....	63
Figure 3-4: Processing in a two-phase commit .....	65
Figure 3-5: Processing in a heuristic decision .....	67
Figure 3-6: Global transaction for a SUP or SPP .....	69
Figure 3-7: Global transaction for an MHP .....	70
Figure 3-8: Transaction control when linked with a J2EE application server .....	72
Figure 3-9: Transaction control when linked with a .NET Framework application .....	73
Figure 3-10: Ranges of the timer monitoring facilities .....	77
Figure 3-11: RPC types .....	82

Figure 3-12: Overview of processing for each RPC type .....	83
Figure 3-13: Comparison of a normal RPC and a chained RPC.....	85
Figure 3-14: Overview of domain-based management.....	87
Figure 3-15: System configuration when using the global search facility.....	92
Figure 3-16: Normal RPC flow .....	95
Figure 3-17: RPC flow when the service information prioritizing function is used.....	97
Figure 3-18: RPC flow when the server UAP has shut down.....	99
Figure 3-19: RPC flow when the server UAP is sustaining a heavy workload .....	100
Figure 3-20: RPC flow when a global search relay node is specified as the priority selection node .....	102
Figure 3-21: RPC flow when no global search relay node is specified as the priority selection node .....	104
Figure 3-22: Example of using the startup notification facility at system switchover.....	107
Figure 3-23: Monitoring other nodes by using the node monitoring facility.....	109
Figure 3-24: Overview of communication using the XATMI interface.....	115
Figure 3-25: Overview of TxRPC communication .....	118
Figure 3-26: Overview of sending and receiving a message by using MCF .....	121
Figure 3-27: Segments in a logical message .....	121
Figure 3-28: Relationship between application program structure and application name.....	122
Figure 3-29: Message processing and MHP application type.....	123
Figure 3-30: Overview of send-only messages.....	124
Figure 3-31: Determination of application name .....	126
Figure 3-32: Overview of ERREVT2 report for a COPNEVT error.....	132
Figure 3-33: Causes of MCF starting a UAP .....	134
Figure 3-34: Message exchange using the input and output queues .....	135
Figure 3-35: Overview of how a UAP receives messages .....	137
Figure 3-36: Overview of how a UAP sends messages .....	139
Figure 3-37: Message sending unified by the dc_mcf_execap() function .....	141
Figure 3-38: Order of sending MCF messages to logical terminals .....	144
Figure 3-39: Sending order based on sending priority combined with FIFO .....	145
Figure 3-40: Scheduling of service requests for SPP.....	147
Figure 3-41: Shutdown of scheduling of service requests to a service group.....	148
Figure 3-42: Shutting down scheduling and storing service requests in the schedule queue	151
Figure 3-43: Maximum number of server processes that can be executed concurrently for a specific service .....	153
Figure 3-44: Maximum number of requests for a specific service that can be placed in the schedule queue.....	154
Figure 3-45: Maximum size of the message-storing buffer pool for a specific service that can be placed in the schedule queue .....	156
Figure 3-46: Scheduling messages for an MHP.....	157
Figure 3-47: Shutting down scheduling by specifying an MCF application name.....	159
Figure 3-48: Overview of Multiserver facility based on resident and non-resident processes.....	164
Figure 3-49: Overview of scheduling priority .....	166



Figure 3-50: Overview of Internode Load-Balancing facility .....	168
Figure 3-51: Scheduling service requests to LEVEL0 nodes .....	173
Figure 3-52: Overview of multi-scheduler facility .....	176
Figure 3-53: Sharing a buffer area .....	178
Figure 3-54: Communication between TP1/Client/W or TP1/Client/P and OpenTP1 .....	184
Figure 3-55: Communication between TP1/Client/J and OpenTP1 .....	185
Figure 3-56: Communication from a server UAP of OpenTP1 to a CUP of TP1/Client/W or TP1/Client/P .....	186
Figure 3-57: TP1/Client communications using MCF message exchange and the TCP/IP protocol.....	188
Figure 3-58: Client/server communications mode when OSI TP is used.....	189
Figure 3-59: Application program communication mode (when OpenTP1 is the client and XDM/DF/TP is the server).....	192
Figure 3-60: Application program communication mode (when XDM/DF/TP is the client and OpenTP1 is the server).....	193
Figure 3-61: Overview of a communication event processing SPP .....	194
Figure 3-62: Remote API facility .....	197
Figure 3-63: Remote procedure call to a UAP inside a firewall.....	200
Figure 3-64: Example of a UAP (SPP) that uses dynamic loading of service functions.....	206
Figure 3-65: Example of a UAP (MHP) that uses dynamic loading of service functions.....	207
Figure 3-66: Example of a UAP (SPP) that uses dynamic loading of service functions.....	208
Figure 3-67: Example of a UAP (MHP) that uses dynamic loading of service functions and a stub .....	209
Figure 3-68: Priorities determining sequence in which resources are used.....	213
Figure 3-69: Example of a deadlock.....	214
Figure 3-70: Acquisition of a user journal .....	215
Figure 3-71: Obtaining a message log .....	218
Figure 3-72: Reception of a message log report .....	220
Figure 3-73: Overview of resource manager control .....	221
Figure 3-74: Start of OpenTP1 and the definitions to be referenced .....	222
Figure 3-75: Overview of the facilities for MCF system statistics .....	223
Figure 3-76: Overview of the real-time statistics service .....	225
Figure 3-77: Concept of using scenario templates for a system operation .....	227
Figure 3-78: Audit logging and main categories of information acquired .....	229
Figure 4-1: Possible locations of an OpenTP1 file system .....	235
Figure 4-2: Selection of files to create the OpenTP1 file system .....	238
Figure 4-3: Separating the OpenTP1 file system into OpenTP1 files and user files on character special files .....	242
Figure 4-4: OpenTP1 file system backup and restore.....	243
Figure 4-5: Swapping of status files .....	248
Figure 4-6: Permitting and prohibiting operation with only one physical file .....	251
Figure 4-7: Filegroup configuration using the parallel access facility (jnl_max_file_dispersion=3) .....	252
Figure 4-8: Overview of the parallel access facility (jnl_max_file_dispersion=3) .....	253

Figure 4-9: Swapping system journal filegroups .....	258
Figure 4-10: Most recent checkpoint dump generation overwrites earlier checkpoint-dump generation .....	261
Figure 4-11: Guaranteed-valid generations and system journal files.....	263
Figure 4-12: Swapping archive journal files.....	272
Figure 4-13: Queue groups and message queue files.....	274
Figure 4-14: DAM file configuration.....	277
Figure 4-15: Extending the block length maintaining the original block configuration.....	281
Figure 4-16: Extending the block length without maintaining the original block configuration.....	282
Figure 4-17: Overview of a cache block chain .....	284
Figure 4-18: TAM tables and TAM files.....	294
Figure 4-19: TAM tables that use large and small hash areas.....	297
Figure 4-20: Example of a deadlock .....	300
Figure 4-21: IST service configuration .....	302
Figure 4-22: Example of effective use of the IST service.....	304
Figure 4-23: How IST records are updated.....	305
Figure 5-1: OpenTP1 environment setup .....	313
Figure 5-2: Environment setup for the message exchange facility .....	314
Figure 5-3: Environment setup for the message queuing facility .....	314
Figure 5-4: Transactions recovered in a complete recovery .....	323
Figure 5-5: User server status at complete recovery after the OpenTP1 system stops during termination processing.....	324
Figure 6-1: Overview of the System Switchover facility.....	352
Figure 6-2: Configuration when using the System Switchover facility .....	354
Figure 6-3: Operation in which a standby system performs only the postprocessing of a running system after a system switchover .....	359
Figure 6-4: Software configuration of OpenTP1 that uses the Multinode facility .....	363
Figure 6-5: OpenTP1 configurations in a cluster system or parallel-processing system .....	365
Figure 6-6: Overview of Global Archive Journal facility .....	368
Figure 6-7: Relation between the global archive journal service and resource groups .....	370
Figure 6-8: A MultiOpenTP1 configuration .....	372
Figure 6-9: OpenTP1 instances and IP addresses (host names) in a one-to-one system switch.....	375
Figure 6-10: OpenTP1 instances and IP addresses (host names) for a double-system switch.....	376
Figure 7-1: How RPCs use ports.....	389
Figure D-1: Overview of processing a remote procedure call to the local node.....	439
Figure D-2: Overview of processing a remote procedure call to remote nodes.....	441
Figure D-3: Overview of global search processing.....	445
Figure D-4: Overview of service information registration processing .....	449
Figure D-5: Overview of service information deletion processing.....	450
Figure D-6: Overview of node-to-node forwarding processing.....	452

Figure D-7: Overview of remote procedure call processing using the dcsvgdef definition  
command.....454

---

## List of tables

---

Table 3-1: List of TP1/Client/J or Client .NET functions supported by the XA resource service.....	75
Table 3-2: Types of timer monitoring and applicable operand .....	78
Table 3-3: XA library subroutines.....	79
Table 3-4: Comparison of node monitoring using the node monitoring facility and the namalivechk command.....	111
Table 3-5: List of MCF events .....	129
Table 3-6: Termination modes and processing of messages in input and output queues.....	143
Table 3-7: Message type and sending priority .....	144
Table 3-8: MCF capabilities that are not supported in Windows.....	146
Table 3-9: Shutting down scheduling of service requests to MCF-applications.....	158
Table 3-10: Shutting down scheduling of service requests to MHP services .....	159
Table 3-11: Shutdown of MHP service group.....	160
Table 3-12: Conditions that determine the load level .....	169
Table 3-13: Operations of the internode load-balancing facility used with other facilities ...	172
Table 3-14: Numbers of remaining service requests and load levels .....	174
Table 3-15: Values specified in user service definition.....	179
Table 3-16: Flow of process control.....	179
Table 3-17: UAPs that can be RAP-processing clients .....	197
Table 3-18: Criteria for relinking UAP objects.....	210
Table 3-19: Lock modes .....	211
Table 3-20: Possibility of sharing resources depending on combination of lock modes .....	212
Table 3-21: Definition of audited events.....	230
Table 4-1: List of OpenTP1 files.....	234
Table 4-2: Differences between the OpenTP1 file system and the OS file system.....	236
Table 4-3: Regular files used in OpenTP1 .....	238
Table 4-4: Example of OpenTP1 file system protection (by owner and access authority)....	244
Table 4-5: Synchronization point journals .....	254
Table 4-6: Recovery journals .....	255
Table 4-7: Statistical journals.....	255
Table 4-8: Differences between when one-system operation is available and when unavailable.....	265
Table 4-9: Special features of DAM files.....	277
Table 4-10: Special features of TAM tables and files .....	294
Table 4-11: Processing when a UAP specifies a key value for TAM table access .....	296
Table 4-12: Loading opportunities and unloading methods for TAM files.....	298
Table 5-1: Routine operations in an OpenTP1 system.....	317
Table 5-2: Operations that modify an OpenTP1 system .....	318
Table 5-3: Other operations in an OpenTP1 system .....	320
Table 5-4: Items monitored by OpenTP1 .....	330

Table 5-5: System definitions related to performance verification trace .....	334
Table 5-6: Relationship between the value specified in the xar_prf_trace_level operand and acquired XAR performance verification trace information .....	335
Table 5-7: Relationship between the jnl_prf_event_trace_level operand value and the trace information that is collected.....	338
Table 5-8: Relationship between the lck_prf_trace_level operand value and the LCK performance verification trace information that is collected .....	340
Table 5-9: System definitions related to the MCF performance verification trace.....	341
Table 5-10: Types of transaction requests and request codes .....	342
Table 5-11: Relationship between the trn_prf_event_trace_condition operand value and the TRN event trace information that is collected .....	344
Table 5-12: Relationship between the value specified in the nam_prf_trace_level operand and acquired NAM event trace information .....	346
Table 6-1: Commands for terminating a running OpenTP1 system .....	360
Table 6-2: Commands for terminating a standby OpenTP1 system .....	361
Table 7-1: System service processes.....	378
Table 7-2: System definition operands that enable you to specify a receive port number .....	388
Table 7-3: Formulas for calculating the number of ports that OpenTP1 uses .....	390
Table 7-4: Information obtainable using the command to check the execution status of temporary closing.....	393
Table A-1: Systems that can be connected to OpenTP1 products that correspond to various communication protocols .....	399
Table B-1: OpenTP1 library functions.....	401
Table B-2: OpenTP1 commands.....	409
Table C-1: Additions and deletions to functions, definitions, and commands in TP1/Server Base 07-03.....	421
Table C-2: Additions and deletions to functions, definitions, and commands in TP1/Message Control 07-03 and TP1/NET/Library 07-04.....	422
Table C-3: Operational changes in TP1/Server Base 07-03 .....	423
Table C-4: Operational changes in TP1/Message Control 07-03 and TP1/NET/Library 07-04.....	424
Table C-5: Changes to defaults in TP1/Server Base 07-03.....	424
Table C-6: Additions and deletions to functions, definitions, and commands in TP1/Server Base 07-02.....	424
Table C-7: Additions and deletions to functions, definitions, and commands in TP1/Message Control 07-02 and TP1/NET/Library 07-03.....	426
Table C-8: Operational changes in TP1/Server Base 07-02 .....	428
Table C-9: Operational changes in TP1/Message Control 07-02 and TP1/NET/Library 07-03.....	429
Table C-10: Changes to defaults in TP1/Server Base 07-02.....	430
Table C-11: Additions and deletions to functions, definitions, and commands in TP1/Server Base 07-01 .....	430
Table C-12: Additions and deletions to functions, definitions, and commands in TP1/Message Control 07-01 and TP1/NET/Library 07-01.....	431

Table C-13: Operational changes in TP1/Server Base 07-01.....	432
Table C-14: Operational changes in TP1/Message Control 07-01 and TP1/NET/Library 07-01 .....	433
Table C-15: Additions and deletions to functions, definitions, and commands in TP1/Server Base 07-00 .....	433
Table C-16: Additions and deletions to functions, definitions, and commands in TP1/Message Control 07-00 and TP1/NET/Library 07-00.....	435
Table C-17: Operational changes in TP1/Server Base 07-00.....	435
Table C-18: Operational changes in TP1/Message Control 07-00 and TP1/NET/Library 07-00.....	436
Table C-19: Changes to defaults in TP1/Message Control 07-00 and TP1/NET/Library 07-00.....	437

## Chapter

---

# 1. Introduction

---

Chapter 1 provides a general overview of OpenTP1 and transaction processing in OpenTP1 systems.

This chapter contains the following sections:

- 1.1 Overview of OpenTP1
- 1.2 Examples of configurations possible with OpenTP1 systems
- 1.3 OpenTP1 software products
- 1.4 OpenTP1 system services

## 1.1 Overview of OpenTP1

OpenTP1 provides online transaction processing<sup>#</sup> in a distributed processing environment.

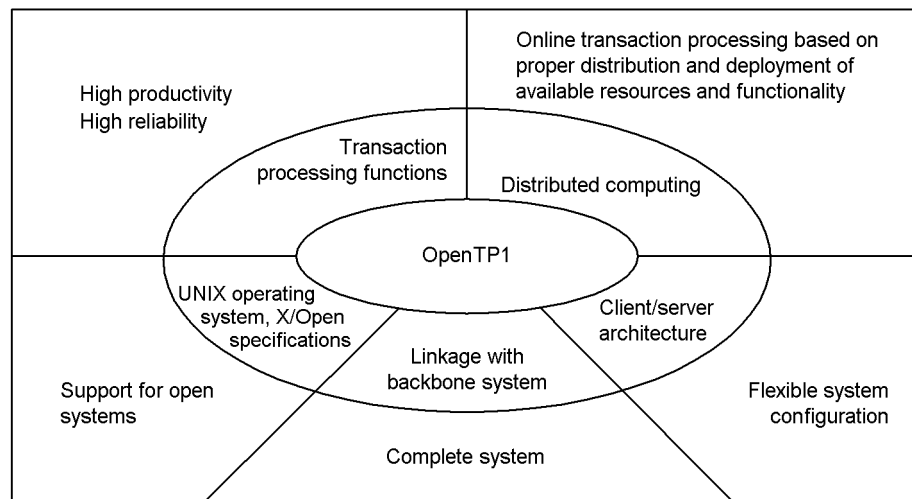
OpenTP1 supports open systems. It complies with the X/Open Distributed Transaction Processing model for distributed processing. OpenTP1 systems can communicate with other open systems.

Organizations can use OpenTP1 to develop and manage online transaction processing systems. These systems can work in OpenTP1 client/server configurations, and can communicate with non-OpenTP1 systems.

During online transaction processing, OpenTP1 can prevent, detect, and recover from errors and failures to a degree that was previously possible on closed mainframe systems or restricted database management systems only.

Figure 1-1 shows the features of OpenTP1.

*Figure 1-1: Features of OpenTP1*



#

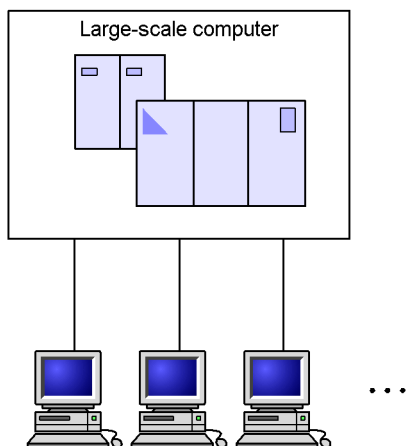
In data communications, the processing must be divided into individual steps and the validity of the result clearly determined. Each step at which the result is judged true or false is known as a *transaction*.



### 1.1.1 Distributed computing environment for transaction processing

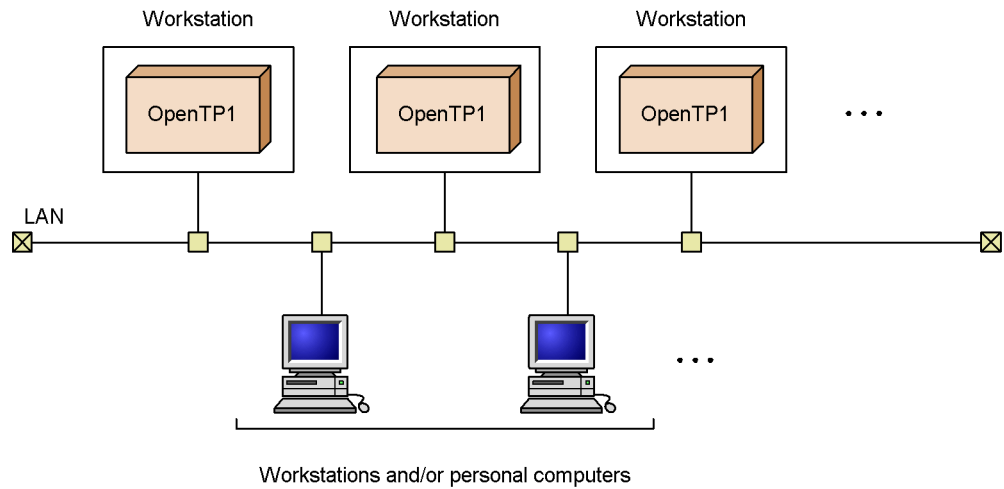
In the past, online transaction processing was often controlled by a large-scale computer that coordinated and performed the processing (Figure 1-2). For example, most bank systems used mainframes to provide centralized facilities for the organizing and processing of transactions.

*Figure 1-2: Conventional online transaction processing in a centralized system using a large-scale computer*



In contrast to the centralized configuration shown above, OpenTP1 provides online transaction processing in a distributed environment. Figure 1-3 shows a possible OpenTP1 configuration.

Figure 1-3: Online transaction processing in a distributed environment using OpenTP1

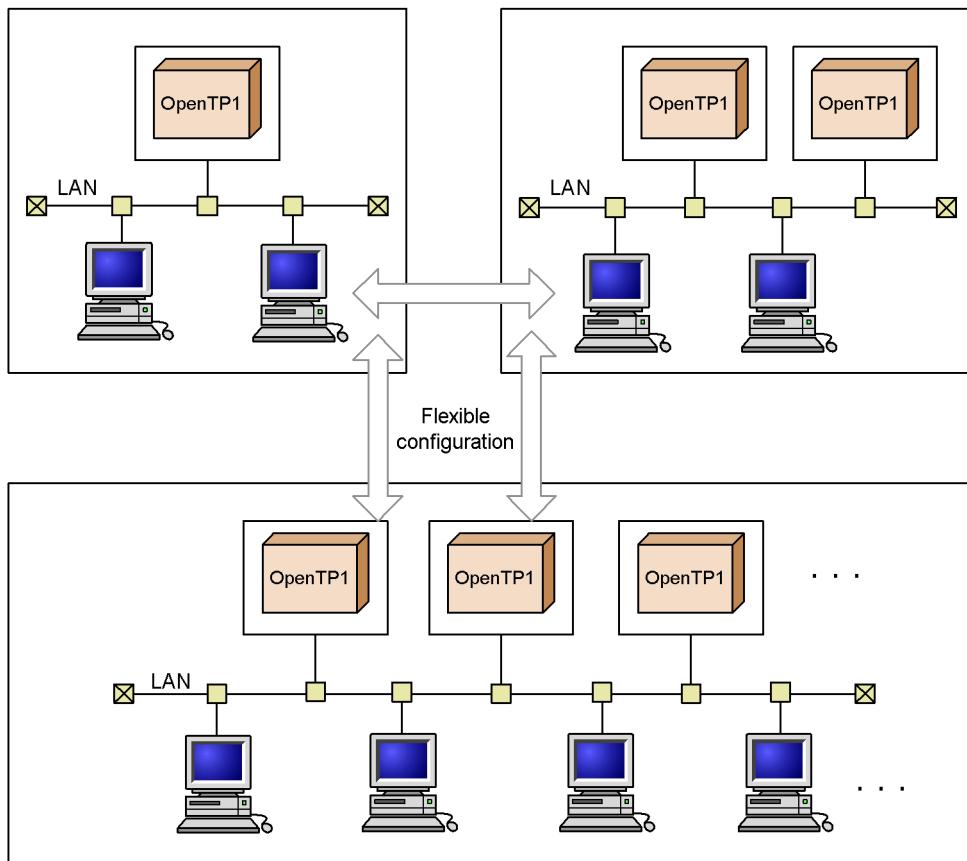


### 1.1.2 Flexible system configuration

To enable online transaction processing in a distributed computing environment, the OpenTP1 system configuration, including *user application programs* (UAPs), is based on the *client/server model*. This allows software resources such as programs and databases to be structured independently of the hardware configuration. Also, if the hardware configuration needs to be expanded to handle increased work loads, there is almost no effect on software resources.

Figure 1-4 illustrates the flexible system configuration.

Figure 1-4: Flexible system configuration



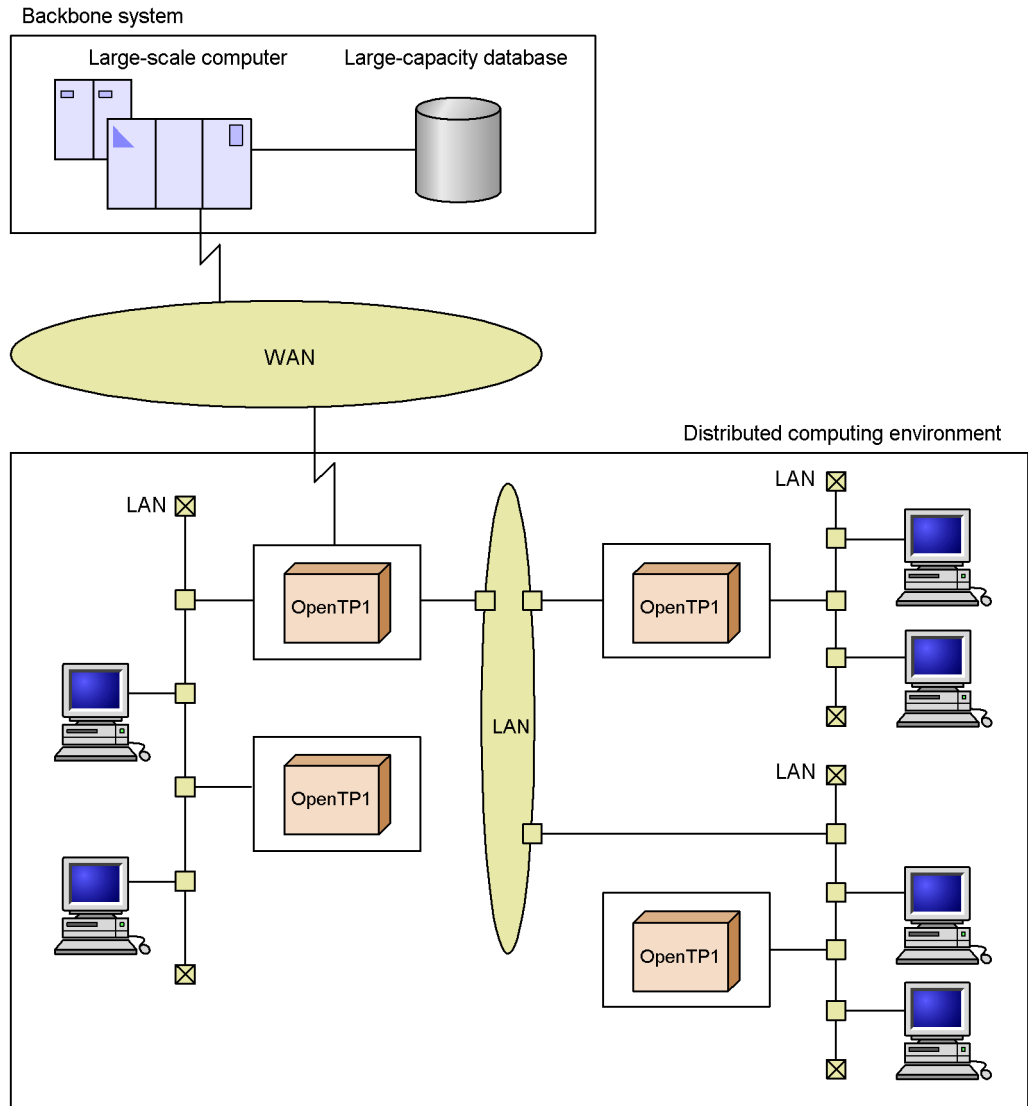
### 1.1.3 Achieving a large-scale system linked to a backbone system

An OpenTP1-based system can be connected to a backbone system such as VOS3. This allows the OpenTP1 distributed computing environment to be integrated with management of the backbone system's large-capacity database and with network management.

OpenTP1 therefore supports the construction of a large-scale system, encompassing the backbone system right down to individual workstations.

Figure 1-5 illustrates a large-scale system incorporating a backbone system.

Figure 1-5: Large-scale system incorporating a backbone system



### 1.1.4 Support for moving to open systems

OpenTP1 complies with the DTP (Distributed Transaction Processing) model specified by the standard-setting organization X/Open. This means that you can write applications to the software interfaces supported by X/Open, and you can use such applications with any software that supports these open standards.

As an example, if a programmer develops a program using the transaction-processing functions specified by X/Open, the program can execute in an OpenTP1 system or any other system that supports the X/Open standards.

### 1.1.5 Unrestricted

In those cases where online transaction processing was not managed by specialized mainframe systems, the transaction processing was usually managed by a relational database manager. Unlike OpenTP1, however, a relational database manager usually manages only those transactions that affect databases controlled by the database manager. OpenTP1 manages transactions that might affect a variety of databases or resources: it is *unrestricted* in its scope and transaction-processing services.

OpenTP1 is a *TP (transaction processing) monitor*. In the computing industry, the term *TP monitor* has started to be used for software that provides full transaction-processing services: that is, software that can manage transactions in a wide variety of systems and is not restricted to managing updates of any single type of database. Unlike the restricted transaction processing services provided by a relational database manager, OpenTP1 provides a full suite of services to support online transaction processing. For example, OpenTP1 provides the means to recover not only from application failures but also from entire system failures. When linked with a relational database management system (RDBMS) OpenTP1 provides a high-speed and high volume data processing server.

OpenTP1 also provides program-development facilities including C, C++, and COBOL support, scheduling facilities, hot-standby systems, and load-balancing facilities.

When coding user application programs (UAPs), you can use C, C++, and COBOL, whichever is convenient for your engineering environment. When coding in C, you can use either the ANSI C, which predates ANSI, or K&R (B. Kernighan and D. Ritchie) format. When coding in COBOL, you can use either COBOL/2 or COBOL85.

You can use the data manipulation language (DML) you usually use with the host computer; so, in OpenTP1 systems you can use a familiar UAP development environment.

---

## 1.2 Examples of configurations possible with OpenTP1 systems

---

An OpenTP1 system can work in a variety of computing configurations. This subsection gives examples of three such configurations:

- a LAN using client/server processing
- an OpenTP1 system connected to a non-OpenTP1 system controlled by a large-scale computer
- a front-end processor

### 1.2.1 OpenTP1 in a LAN that uses client/server processing

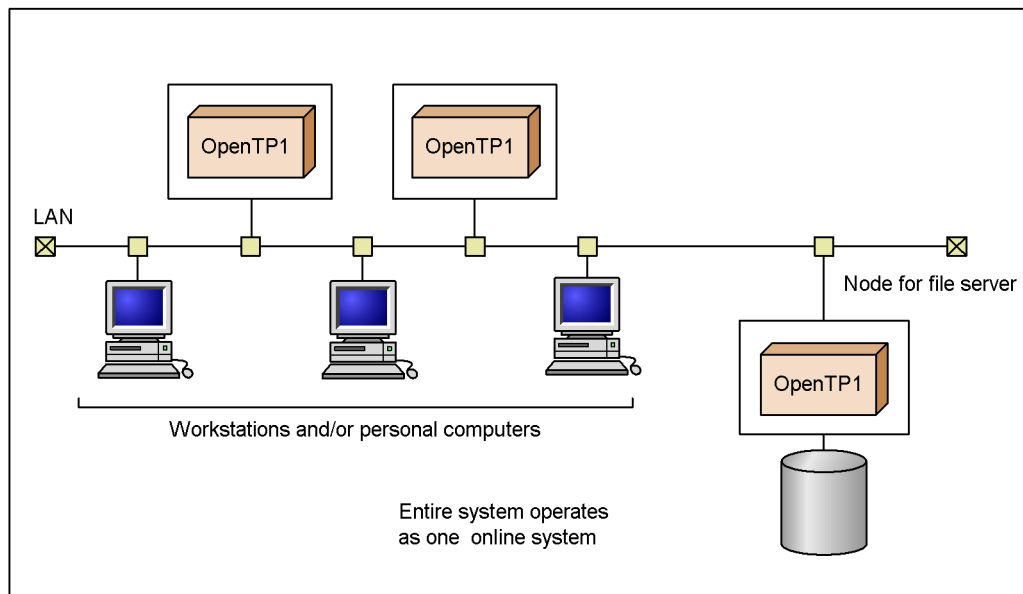
OpenTP1 can be used in LANs in which distributed applications are constructed using the client/server model. In such a system a server application can be coded so that it can provide a service to clients located on other workstations. Client application code contains special OpenTP1 functions that indicate the start and end of a transaction, and OpenTP1 manages such a transaction even if parts of the transaction are processed on different machines in the system.

When such a system needs to be expanded to handle increased work loads, the required hardware can be added while continuing to use the online transaction processing system and without complicated reorganizations of software resources. In this way you can flexibly construct systems that are appropriate for the required work situation.

Such LAN systems also enable collaboration between servers, and enable full use of advanced GUI applications.

Figure 1-6 illustrates an OpenTP1 online transaction processing system in a LAN. Note that the diagram contains the word *node*. In this manual, a node is a computer (machine) on which OpenTP1 operates and which is connected to a network.

Figure 1-6: OpenTP1 online transaction processing using the client/server model in a LAN



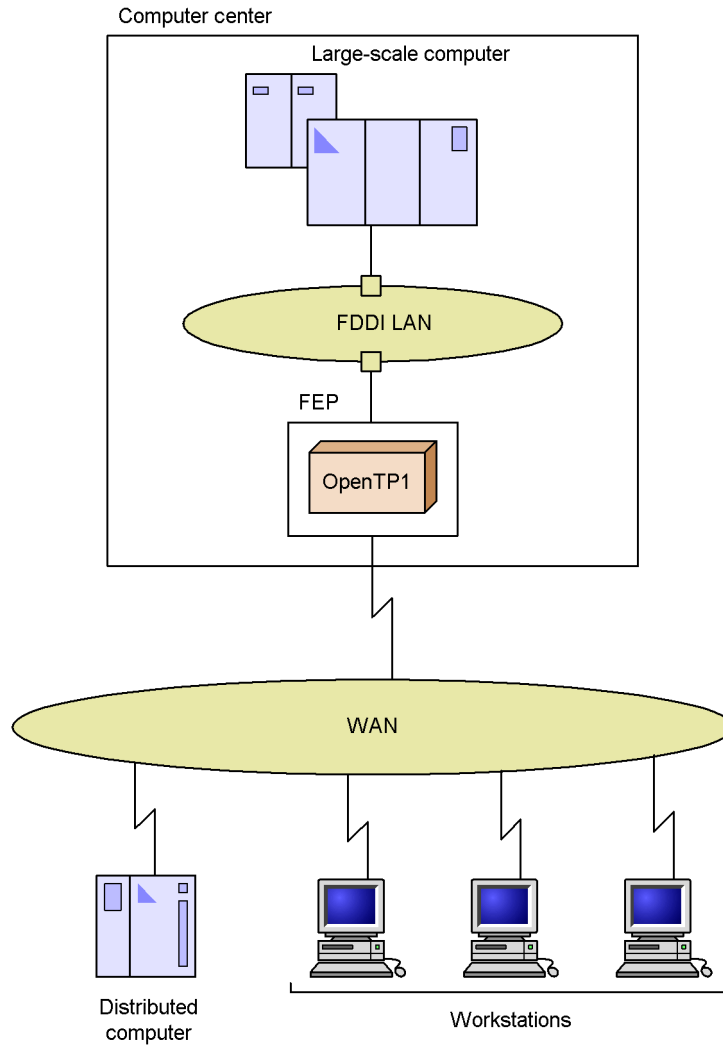
### 1.2.2 OpenTP1 in front-end processors

OpenTP1 can also be installed on a FEP (front-end processor) connected to a large-scale computer. A FEP acts as an intermediary between the large-scale computer and terminals.

Using OpenTP1 on the FEP provides both communication and failure-recovery benefits. OpenTP1 supports multiple communication protocols, so OpenTP1 enables the FEP to handle multi-protocol and multi-line connections over a WAN. Also the error-recovery features of OpenTP1 improve reliability. For example, OpenTP1 can take over transaction processing in case of a failure in the host computer.

Figure 1-7 shows an example of how an FEP is configured.

Figure 1-7: Example of an front-end processor (FEP)



### 1.2.3 OpenTP1 connected to a non-OpenTP1 system

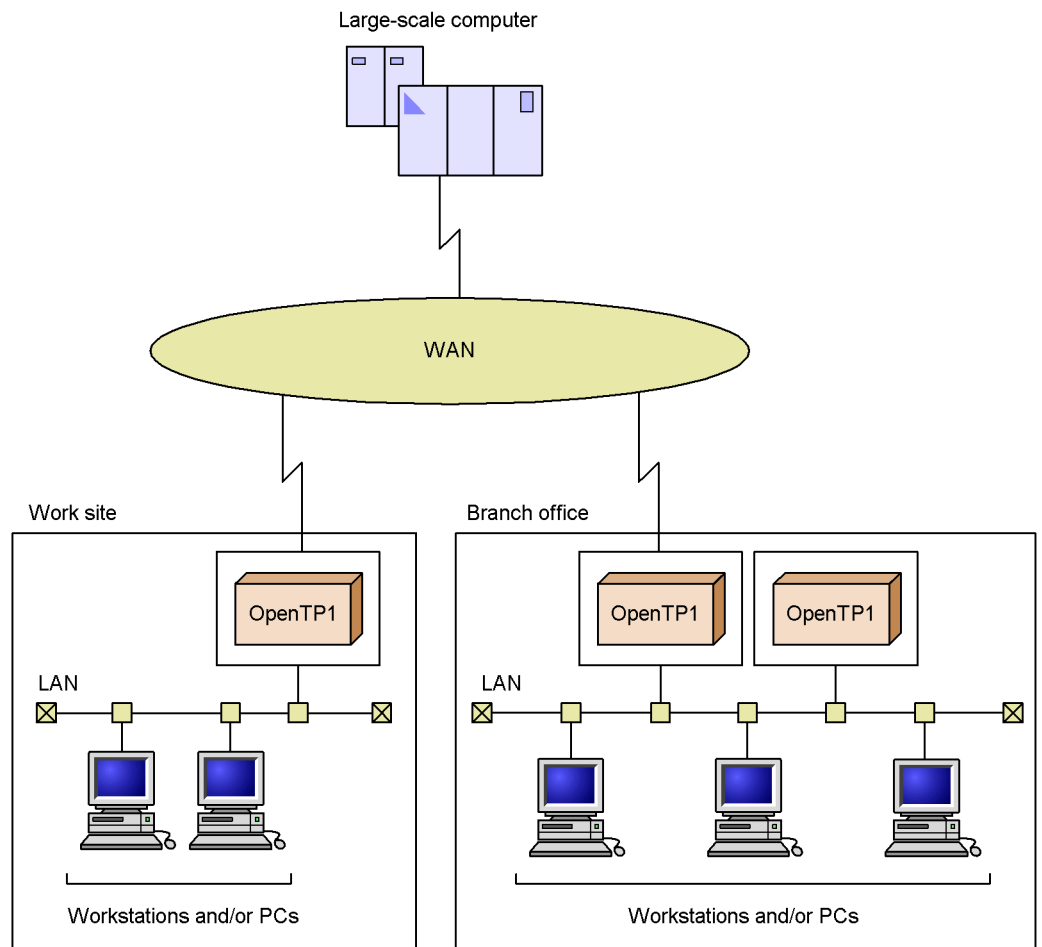
OpenTP1 enables transaction processing even in distributed systems where some of the required data or applications are contained in a non-OpenTP1 system, such as a large-scale computer. For example, an organization might have a mainframe connected via a WAN to LANs at work sites and branch offices (Figure 1-8). In this configuration, a workstation or PC can use *messages* to exchange data with the host computer. A message can send data to the large-scale computer for processing, and the results can be returned by another message. This process can be managed as a transaction



controlled by OpenTP1.

Figure 1-8 shows an example of how a distributed system can be configured.

*Figure 1-8: OpenTP1 online transaction processing in a system using a large-scale host computer*

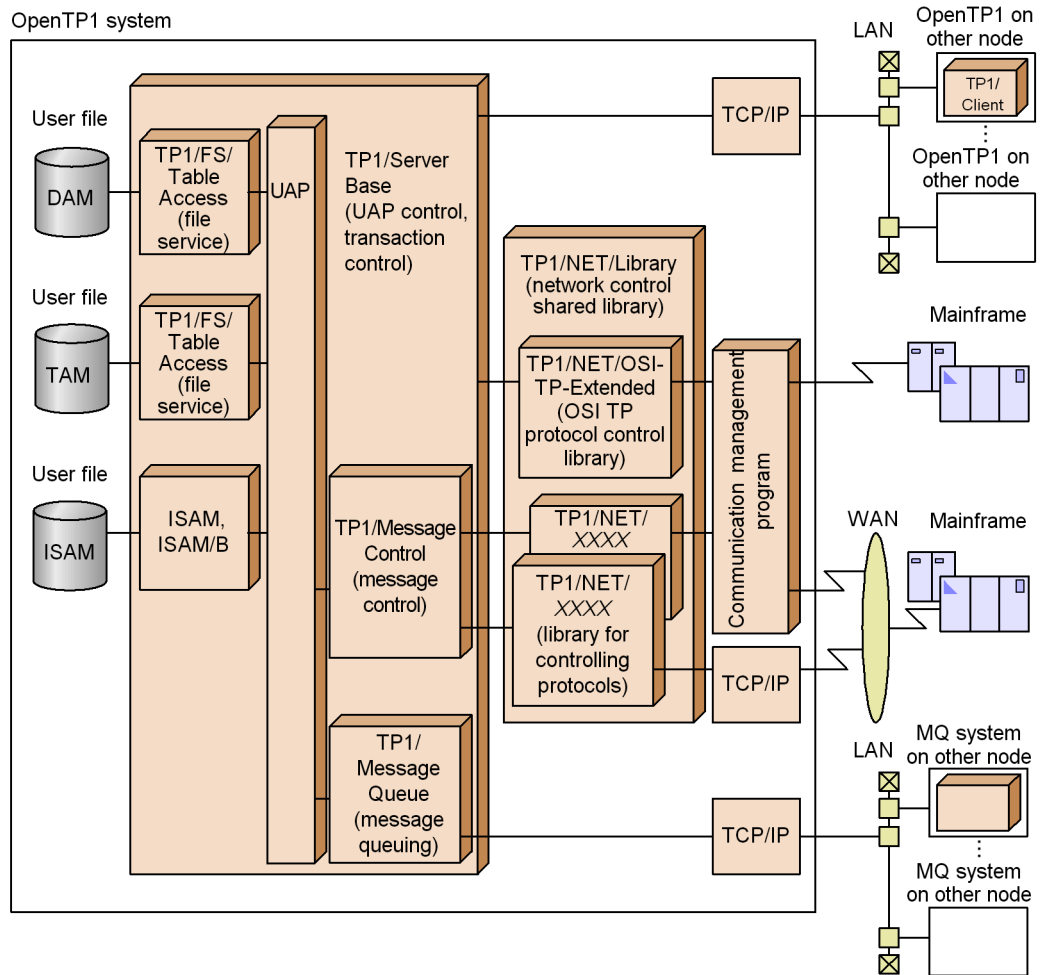


## 1.3 OpenTP1 software products

This section gives an overview of the various OpenTP1 software products.

Figure 1-9 illustrates how the OpenTP1 software products listed in the previous section interact.

Figure 1-9: OpenTP1 software products



### 1.3.1 List of OpenTP1 software products

The OpenTP1 software system is made up of a number of software components or *software products*. OpenTP1 administrators and programmers should have a general

understanding of which products are performing which functions in an OpenTP1 system. A system administrator needs such knowledge when deciding which OpenTP1 components should be installed in a node. A programmer needs such knowledge when deciding what program products must be present in order for his or her program to work.

The following subsections list and briefly describe the OpenTP1 software products.

### **(1) Basic components**

The following software products provide the basic features of OpenTP1:

TP1/Server Base

Performs fundamental control of distributed transaction processing. This product provides facilities for controlling transactions and scheduling UAPs

### **(2) User data management**

The following products provide user data management:

TP1/FS/Direct Access

Enables OpenTP1-dedicated user files to be accessed by the direct access method (DAM): such direct-organization files are called DAM files. The TP1/FS/Direct Access product manages file status so that DAM file updates can be made part of an OpenTP1 transaction.

TP1/FS/Table Access

Enables OpenTP1-dedicated user files to be accessed by the table access method (TAM): such files are loaded into memory in table format and are called TAM files. OpenTP1 TAM files enable fast access. The TP1/FS/Table Access product manages table status so that the in-memory information is recoverable and TAM file updates can be made part of an OpenTP1 transaction.

TP1/Shared Table Access

The IST service allows information from all nodes in a group to be shared among the nodes and to be accessed from any node in the group. The information is stored in special tables called *internode shared tables* (ISTs). The IST service stores internode shared tables in shared memory, and enables UAPs to reference and update the tables without knowing the actual physical locations of the information that makes up the tables.

### **(3) Message control facilities (MCF)**

The following products provide the facilities for sending and receiving messages:

TP1/Message Control

Used when an OpenTP1 system communicates with a non-OpenTP1 system. For example, the non-OpenTP1 system might be in a non-UNIX network connected

to the OpenTP1 system via a WAN. The TP1/Message Control product uses MCF message queues to control the sending and receiving of messages.

TP1/NET/Library

Provides a library that contains the facilities for operations required to control networks, for configuration management, and for scheduling.

TP1/Message Control and TP1/NET/Library make up the message-exchange configurations.

TP1/NET/xxx

In addition to TP1/Message Control and TP1/NET/Library, a product corresponding to the communication protocol in the non-OpenTP1 system is also required to communicate with the non-OpenTP1 system.

**(4) Message queue access (MQA)**

The following product is necessary for the MQA message queuing of OpenTP1:

TP1/Message Queue

Used when transferring messages asynchronously to a UAP of OpenTP1 or a UAP of another system (via a network). Messages can be transferred regardless of the status of the destination UAP without waiting for the response of the destination UAP.

**(5) System switching**

The following products provide the System Switchover facility, which enables the construction of hot-standby systems. The products are usable only when the OS is HI UX/WE2.

TP1/High Availability

Used in system configurations that provide the System Switchover facility.

TP1/NET/High Availability

Used when the message control facilities (the TP1/Message Control and TP1/NET/Library products) are used in system configurations that contain the System Switchover facility. The TP1/NET/High Availability product requires TP1/High Availability and HAMonitor.

The TP1/NET/High Availability product is also used when changing connections during message transmission with duplicated systems.

**(6) Testing user application programs**

The following products are used for testing user-created programs that work in an OpenTP1 system.

TP1/Offline Tester

Provides the Offline Tester. This Offline Tester makes it possible to test user applications in environments in which OpenTP1 is not running.

TP1/Online Tester

Provides the Online Tester. This Online Tester makes it possible to test user applications in OpenTP1 online environments.

TP1/Message Control/Tester

Provides the MCF Online Tester. This MCF Online Tester makes it possible to test various OpenTP1 message-sending and message-receiving facilities. This product is usable only when the OS is HI-UX/WE2.

**(7) Products for client machines**

A UAP can function as a client when the following products are installed on the same machine as the UAP.

TP1/Client/W and TP1/Client/P

Used when a workstation or PC is used as a client machine and the machine is used to access a server constructed in an OpenTP1 system.

TP1/Client/J

Used when you use a Java applet, a Java application, or a Java servlet to access a server constructed in an OpenTP1 system.

For details on TP1/Client/W, TP1/Client/P, and TP1/Client/J, see 2.2.2 *Using OpenTP1 client software on workstations and PCs*.

**(8) Distributed application server facility**

The following product provides the facilities for distributed application servers:

TP1/LiNK

Provides basic control for UAP scheduling in a distributed environment. Compared to TP1/Server Base, TP1/LiNK is more applicable to small-scale departments. Operations from installation to setup of TP1/LiNK can easily be performed interactively.

**(9) Cluster system or parallel-processing system**

The following product is an operation aid in a cluster system or parallel-processing system:

TP1/Multi

Aids operations when using OpenTP1 in a cluster system or parallel-processing system environment. This product makes it possible to manage multiple nodes from a single node.

### **(10) Controlling resource managers not provided by OpenTP1**

The following product is required for controlling resource managers that are not provided by OpenTP1.

TP1/Resource Manager Monitor

Enables you to control any resource managers that are not provided by OpenTP1. TP1/Resource Manager Monitor allows OpenTP1 to control the startup and termination of resource managers.

### **(11) Related products**

The following Hitachi products work with OpenTP1.

ISAM, ISAM/B (HP-UX only)

Makes it possible to use indexed sequential files that comply with the X/Open ISAM (indexed sequential access method) model. When ISAM user files are managed as part of transaction processing, ISAM/B is required in addition to Hitachi ISAM.

HAMonitor

Used when improving the reliability of OpenTP1 systems. Reliability can be improved by duplicating OpenTP1 systems and using the System Switchover facility. For details on OpenTP1 systems using HAMonitor, see *6.1 The System Switchover facility*.

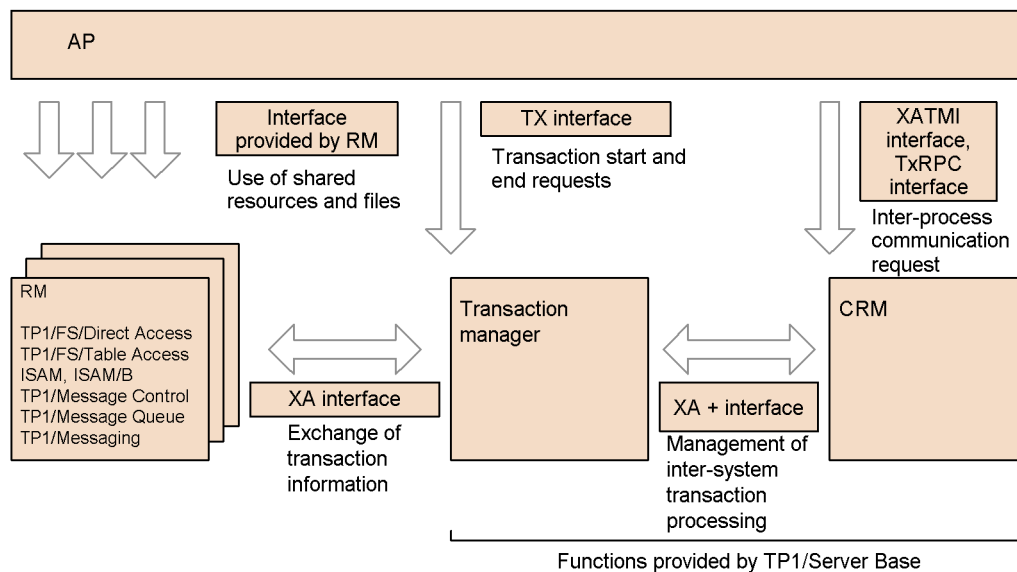
## **1.3.2 OpenTP1 and the X/Open DTP model**

In the X/Open DTP model, transaction processing is an interaction among:

- a UAP (user application program)
- a transaction manager
- a resource manager
- a communication resource manager
- interfaces among the UAP, TM, RM, and CRM

The *UAP* specifies the contents of the transaction and directs a transaction manager to start or terminate a transaction. A *transaction manager* manages the transaction and coordinates the completion of a transaction across a set of resource managers. A *resource manager* manages a particular shared resource. A *communication resource manager* manages inter-application components. Figure 1-10 illustrates the relation among the components of the model and their interfaces.

Figure 1-10: The X/Open DTP model



### (1) Configuration of DTP models

The DTP models are composed of the following elements.

**AP** (Application Program)

An application program created by the system user using a high-level language. This application program is known as a UAP of OpenTP1.

**Transaction Manager**

Manages system transactions and supports consistency according to the update information of resources.

**RM** (Resource Manager)

Manages system resources such as user data.

**CRM** (Communication Resource Manager)

Manages resources concerning the communication between the systems.

### (2) Interface between elements

Each element that composes a DTP model can be linked with the following interfaces.

**TX interface**

Instructs the start and end of a transaction from AP to Transaction Manager.

**XATMI and TxRPC interfaces**

Instruct the communication from AP to CRM.

XA interface

Synchronizes according to the update information of resources with Transaction Manager and RM.

XA+ interface

Extends the transaction managed by Transaction Manager to other system processing when communicating with other system using CRM.

Interface provided by RM

Instructs update of resources from AP to RM. The API to RM contains an SQL.

### **(3) Resource managers provided by OpenTP1**

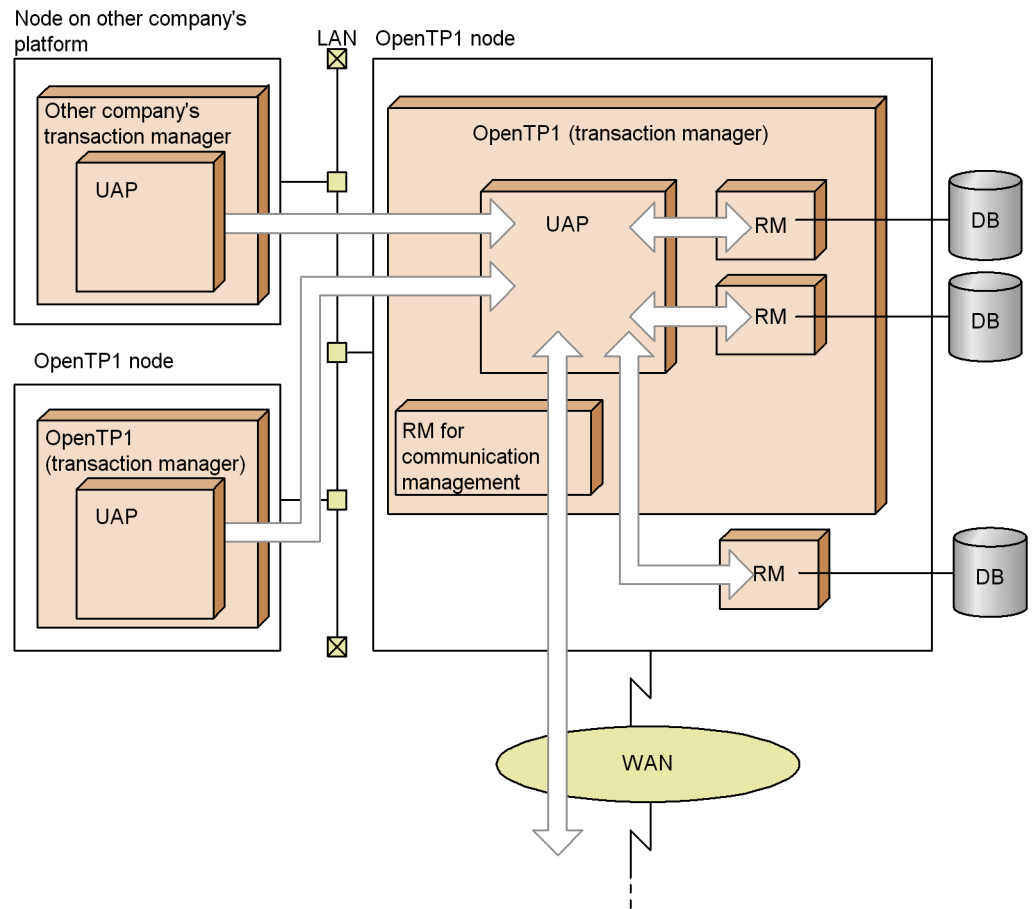
The following shows RMs that can be used in OpenTP1.

- RMs that manage user resources such as files  
An RM that conforms to TP1/FS/Direct Access, TP1/FS/Table Access, ISAM (ISAM/B), and X/Open
- RMs that manage communications by using messages
- TP1/Message Control, TP1/NET/Library, and TP1/Message Queue.

Figure 1-11 shows the relationship between the X/Open DTP model and the OpenTP1 system.



Figure 1-11: Relationship between the X/Open DTP model and the OpenTP1 system



---

## 1.4 OpenTP1 system services

---

### 1.4.1 Types of OpenTP1 services

Various service functions can be used in an OpenTP1 system to build an online transaction processing system based on the client/server model. A *service* is a program unit that can be called from another process; the process that provides the service is called a *server*.

OpenTP1 services are known as *system services*. The server that provides job processing as a service, based on a UAP created by the user, is known as a *user server*.

### 1.4.2 OpenTP1 system services

The system services provided by OpenTP1 are described below. The three letters in parentheses are the service identifier.

#### **(1) General system services provided by the OpenTP1 system**

Transaction service (trn)

Starts and terminates transaction processing, and manages the processing for updating resources based on reaching a synchronization point in a transaction.

Name service (nam)

Manages the correspondences between LAN network addresses and user servers in order to provide inter-UAP communications (remote procedure calls).

Schedule service (scd)

Manages the scheduling for remote procedure calls and for message control facilities.

Process service (prc)

Manages processes within a node.

Status service (sts)

Mainly stores history information unrelated to transactions into dedicated files, and manages the information.

Interval service (itv)

Manages the monitoring of internal processing times for each system service listed here.

Journal service (jnl)

Stores history information (journal) related to transaction processing in system journal files and checkpoint dump files, and manages the information.

**Checkpoint dump service (cpd)**

Sets and manages checkpoints in the system journal file to reduce the time for reading journals.

**Lock service (lck)**

Manages various locks in accordance with the transaction processing.

**Log service (log)**

Manages the message log that an OpenTP1 system uses to inform an operator of internal information.

**Time service (tim)**

Provides various time-monitoring facilities such as monitoring whether a transaction is taking too much time.

**(2) System services that manage user data****DAM service (dam)**

Manages OpenTP1-dedicated user files (DAM access) that are used in application processing.

**TAM service (tam)**

Manages OpenTP1-dedicated user files (TAM access) that are used in application processing.

**IST service (ist)**

Manages the internode shared tables in memory shared by multiple OpenTP1 systems.

**(3) System services related to Message Control facility****MCF service (MCF manager service, MCF communication service, application startup service)**

Provides various message-control facilities for communicating with non-OpenTP1 systems.

**MCF message queue service (que)**

Manages the wait queues (input and output queues) that store messages when using the message control facilities.

**Mapping service (map)**

Aid for using, in OpenTP1, the Extended Presentation facility for GUIs.

These services require TP1/Message Control, which is the resource manager for message control.

**(4) System services related to Message Queuing facility**

MQA service (mqa)

Manages MQA message queuing when transferring messages between UAPs asynchronously. This service requires TP1/Message Queue, which is the resource manager for controlling MQA message queuing.

**(5) System service that manages resource managers**

Resource manager monitor service (rmm)

Controls the start and termination of resource managers via OpenTP1.

**(6) System service that manages the processes used by the OpenTP1 client facility (TP1/Client)**

Client service (clt)

Manages the processes that are required for TP1/Server Base when starting a transaction from an application program (CUP) of TP1/Client.

**(7) System service that manages the online tester (TP1/Online Tester)**

Tester service (uto)

Manages the execution environment of the online tester.

**(8) System service that provides help**

Transaction journal service (tjl)

Stores history information (journal) of transaction processing as a dedicated journal (the transaction recovery journal), and manages the information and journal. A user might decide to obtain such history information for transactions that take a long time in order, for example, to reduce recovery time.

Real-time statistics service (rts)

Manages real-time statistics that are required to check the operating status of the OpenTP1 system in real time.

**(9) System service that manages archive journal files in a multinode environment**

Global archive journal service (jnl)

Archives and manages the system journal files for each OpenTP1 node in a cluster system or parallel-processing system.

**(10) System service that controls indexed sequential files that comply with the X/Open ISAM standard**

ISAM service (ism)

Manages indexed sequential user files (ISAM) that comply with X/Open ISAM

standards and are used in application processing.

### 1.4.3 OpenTP1 system definitions

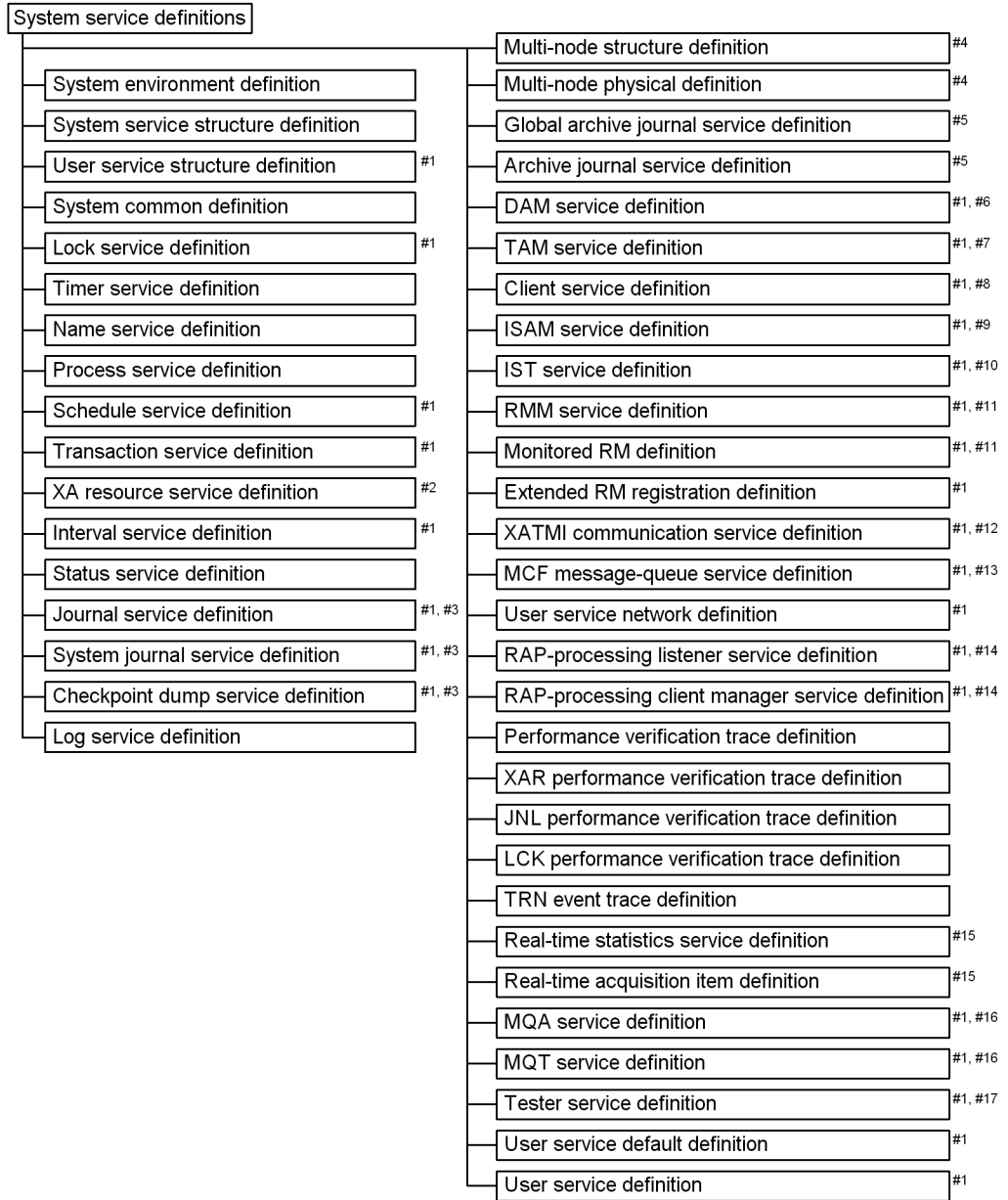
The OpenTP1 operating environment and the resources used by OpenTP1 are specified in *system definitions*. The OpenTP1 system definitions are categorized as follows:

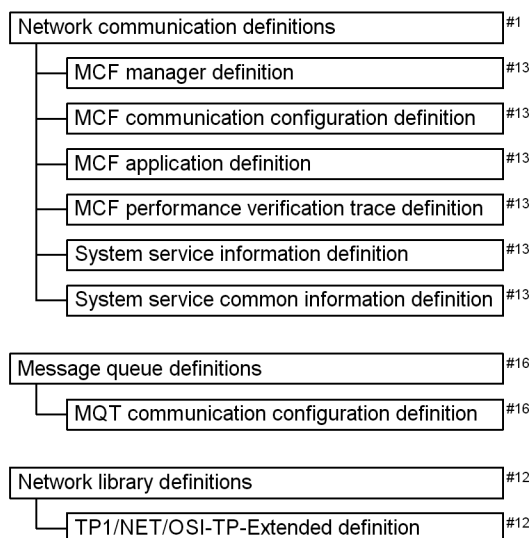
- System service definitions  
Defines items relating to the basic facilities of OpenTP1 (TP1/Server Base), file services, the multi-node facility, and the message queuing facility.
- Network communication definitions  
Required when using the message exchange facility (TP1/Message Control).
- Message queue definitions  
Required when using the message queuing facility (TP1/Message Queue).

To create these definitions, prepare a file using a text editor. For details about the system definitions, see the manual *OpenTP1 System Definition*.

Figure1-12 shows how the OpenTP1 system definitions are organized.

Figure 1-12: Organization of OpenTP1 system definitions





- #1: Not required for an OpenTP1 node that uses the global archive journal service.
- #2: Required when using the XA resource service function.
- #3: Not required for an OpenTP1 node that uses the journal fileless mode.
- #4: Required when using TP1/Multi.
- #5: Required when using the global archive journal service.
- #6: Required when using TP1/FS/Direct Access.
- #7: Required when using TP1/FS/Table Access.
- #8: Required when using TP1/Client/W or TP1/Client/P.
- #9: Required when using the facility for transaction process of ISAM files (ISAM/B). For details about the definitions, see the manual *Indexed Sequential Access Method ISAM*.
- #10: Required when using TP1/Shared Table Access.
- #11: Required when using TP1/Resource Manager Monitor.
- #12: Required when performing OSI TP communications using TP1/NET/OSI-TP-Extended. For details about the definitions, see the manual *OpenTP1 Protocol TP1/NET/OSI-TP-Extended*.
- #13: Required when using the message exchange facility (TP1/Message Control).
- #14: Required when using the remote API facility.
- #15: Required when using the real-time statistics service.
- #16: Required when using the message queuing facility (TP1/Message Queue). For details about the definitions, see the *OpenTP1 TP1/Message Queue User's Guide*.
- #17: Required when using TP1/Online Tester. For details about the definitions, see the *OpenTP1 Tester and UAP Trace User's Guide*.





## Chapter

---

# 2. Application Processing Modes

---

This chapter describes the modes of application processing that can be performed in OpenTP1, related products, and the types of user application programs used in an OpenTP1 system.

- 2.1 Overview of OpenTP1 communications
- 2.2 Processing in a client/server configuration
- 2.3 Processing in an MCF message-exchange configuration
- 2.4 Processing in an MQA message-queuing configuration
- 2.5 Other Hitachi software products usable with OpenTP1
- 2.6 User application programs in OpenTP1 systems
- 2.7 Processing in an Internet-based configuration

---

## 2.1 Overview of OpenTP1 communications

---

This section gives an overview of the three major types of OpenTP1 computer configurations:

- client/server configurations that use remote procedure calls for communication
- MCF message-exchange configurations
- MQA message-queuing configurations
- Overview of client/server configurations and remote procedure calls

In a client/server configuration, a client application makes a request to a server application to perform some processing, and the server returns the results of the processing to the client. In OpenTP1 systems, a client/server configuration is usually in a LAN in which the UAPs (user application programs) use the TCP/IP protocol to communicate via remote procedure calls

- Overview of MCF message-exchange configurations

An MCF message-exchange configuration enables messages to be exchanged between OpenTP1 and non-OpenTP1 systems using communication protocols other than (or including) the TCP/IP communications protocol. For example, a non-OpenTP1 system might be a network controlled by a large-scale host computer, and the communications protocol is the protocol used in that network. A UNIX workstation could use the OpenTP1 message facilities to send data to be processed on the large-scale host computer, and the host can use messages to send processed data back to the workstation. OpenTP1 provides compatibility with a number of communications protocols.

- Overview of MQA message-queuing configurations

In communication that uses MQA message queuing, messages can be exchanged between systems that have queue managers. OpenTP1 uses the TCP/IP protocol to support the communication using MQA message queuing. OpenTP1, XDM, and MQSeries all use queue managers. For further details, see *2.4 Processing in an MQA message-queuing configuration*.

---

## 2.2 Processing in a client/server configuration

---

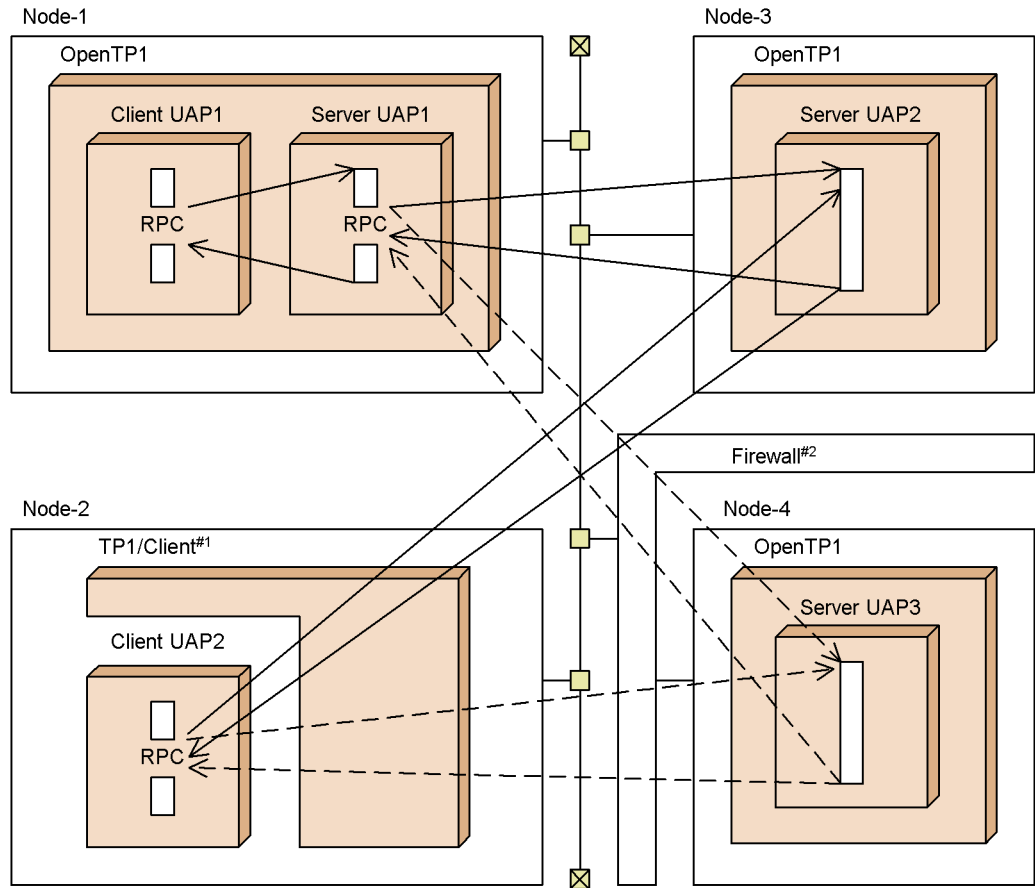
This section describes communication in a client/server configuration. In this type of configuration, the protocol used for communicating with another UAP process is transparent to the UAP.

### 2.2.1 Communication via remote procedure calls

The UAPs in an OpenTP1 system can communicate with other UAPs using *remote procedure calls* (RPCs). A UAP process uses an RPC to request a service from another UAP process, and the UAP process that received the request uses an RPC to return the processing result to the requesting UAP.

Figure 2-1 shows how processing is performed in a client/server configuration.

Figure 2-1: Overview of communication in a client/server configuration



#1

TP1/Client is a client-specific product that requests services from an OpenTP1 server.

For details about TP1/Client, see 2.2.2 *Using OpenTP1 client software on workstations and PCs* and 3.5 *OpenTP1 client facility (TP1/Client)*.

#2

With the OpenTP1 interface, RPCs can be used to communicate with a server UAP behind a firewall. However, when requests cross a firewall, the server UAP cannot be included in the transaction processing. For details about communicating with a UAP behind a firewall, see 3.7.1 *Example of using the remote API facility*.

**(1) Client UAP and server UAP**

The UAPs that request and provide services, respectively, are in a client/server relationship. The UAP on the requesting side is referred to as a *client UAP* and the UAP on the service-providing side is referred to as a *server UAP*.

For details about UAPs, see *2.6 User application programs in OpenTP1 systems*.

**(2) Transactional RPCs**

OpenTP1 also provides programmers with functions that cause OpenTP1 to treat an RPC as a transaction. For example, coding `dc_trn_begin()` before `dc_rpc_call()` will cause OpenTP1 to handle the processing following the `dc_rpc_call()` call as a transaction. In such a case `dc_rpc_call()` is said to be a *transactional RPC*. The transaction extends over the period from when a service is requested to when the results are returned. OpenTP1 can handle transactional RPCs that cause processing over multiple nodes.

**(3) Remote procedure calls supported by OpenTP1**

The following describes the OpenTP1 interface provided for client/server communications.

**(a) OpenTP1 interface**

The OpenTP1 interface is a communication method that enables services to be requested using OpenTP1 library functions.

**(b) Compatibility with the X/Open interface**

In addition to the native OpenTP1 functions, OpenTP1 also supports remote procedure calls and functions compatible with the X/Open interface. Programmers can thus code OpenTP1 client/server applications by writing applications using the standard X/Open interface. Communication is possible via the XATMI and TxRPC interfaces.

For details about client/server communications, see *3.2 Processing in an OpenTP1 client/server configuration*.

**(4) Client/server configuration communication protocols**

For the client/server configuration communication protocol of OpenTP1, *TCP/IP* can be used. You need not to be aware of the communication protocol in UAP processing.

**(5) TCP\_NODELAY**

OTenTP1 provides functionality that guarantees no-delay data transmission even during a wait for a response to data already sent.<sup>#</sup> This functionality can be enabled by using the `TCP_NODELAY` option for the socket (INET domain) that OpenTP1 uses for inter-node communication.

You can specify whether to use the `TCP_NODELAY` option for data transmission by using the `ipc_tcpnodelay` operand in the schedule service definition, the user

service definition, or the user service default definition. Note that if this option is used, the efficiency of sending data in INET domain communication may be degraded and the network load may increase. Before using the option, carefully consider whether the option is necessary by taking into account the `ipc_sendbuf_size` operand, the `ipc_recvbuf_size` operand, the network bandwidth, and other factors.

#

This functionality is implemented by disabling the Nagle algorithm in TCP/IP.

### 2.2.2 Using OpenTP1 client software on workstations and PCs

The following OpenTP1 program product enables UAPs on workstations or PCs to act as clients and request services from OpenTP1 server systems:

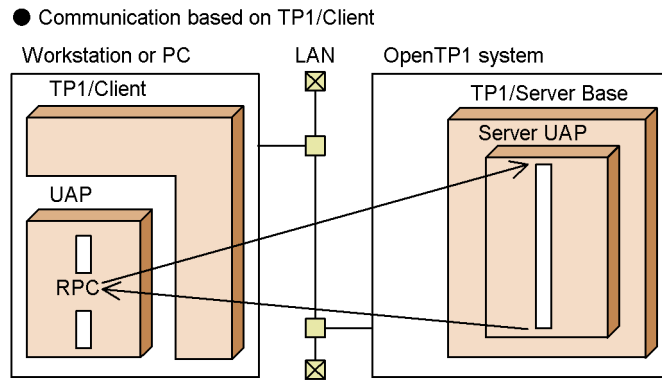
TP1/Client

Enables UAPs on PCs or workstations to request and receive services from OpenTP1 server UAPs. The UAP of TP1/Client/W and TP1/Client/P are called *CUP*.

TP1/Client UAPs (CUPs) can communicate with OpenTP1 server UAPs (SPPs). TP1/Client actually contains four products:

- TP1/Client/W for workstations
- TP1/Client/P for PCs
- TP1/Client/J for Java-operating environments
- These products enable a system manager to select client hardware and the OS (a workstation or PC) according to the expected performance, number of connections, and the OS required for the client.

Figure 2-2 provides an overview of the communication processing that can be performed with the OpenTP1 client facility.

*Figure 2-2: Overview of communication based on TP1/Client*

For details about TP1/Client, see 3.5 *OpenTP1 client facility (TP1/Client)*.

---

## 2.3 Processing in an MCF message-exchange configuration

---

This section gives a general description of MCF message exchange. *MCF message exchange* refers to sending and receiving messages by using some communications protocol. MCF message exchange is convenient for communicating between OpenTP1 systems and non-OpenTP1 systems.

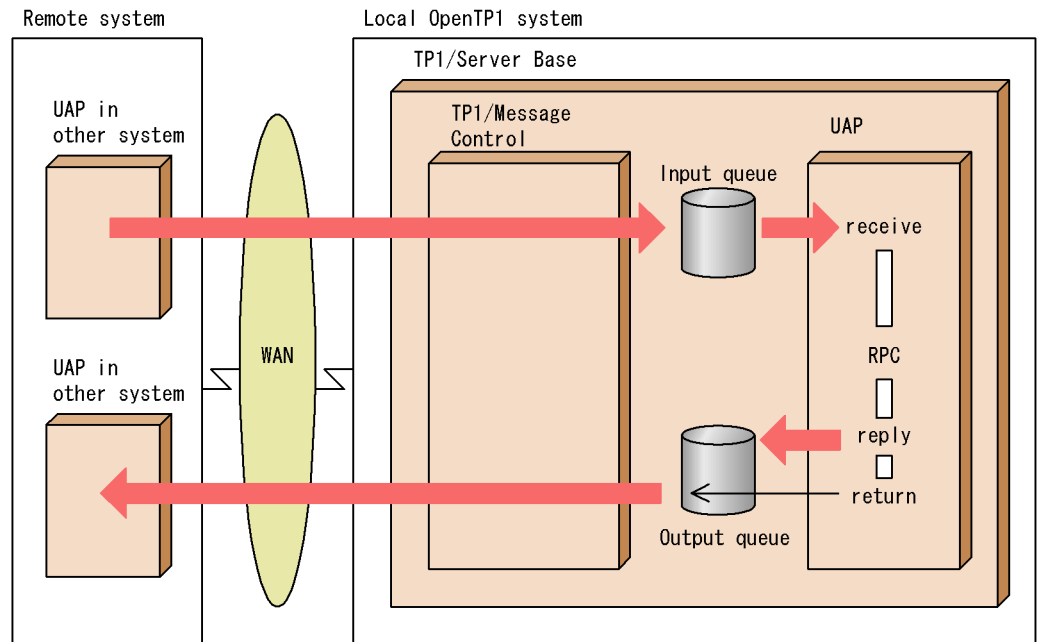
### 2.3.1 Overview of MCF message exchange

In MCF message exchange, *MCF message queues* are used to store messages. In an OpenTP1 system when an MCF message is received from another system, MCF temporarily stores the received message in an *input queue*. MCF then activates a UAP, called a *message-handling program*, to process the message. The message-handling program receives the message (with the `dc_mcf_receive()` function) and processes it. After the message is processed, the message-handling program uses the reply function (`dc_mcf_reply()`) and MCF temporarily stores the reply message in the *output queue*. When the message-handling program terminates normally (e.g., a C program executes `return`), MCF sends the messages stored in the output queue to the other system.

Figure 2-3 gives an overview of the processing in sending and receiving MCF messages. The large arrow indicates the data flow of messages. The black arrow indicates how execution of `return` in a C program causes MCF to send the MCF message from the output queue to the other system.



Figure 2-3: Overview of MCF message-exchange processing



### 2.3.2 Networks that can use MCF message exchange

MCF message exchange is mainly used in a WAN (wide-area network) for communications between a system that contains OpenTP1 and a system that lacks OpenTP1. For example, the non-OpenTP1 system might be controlled by a large-scale host computer. Communications with non-OpenTP1 systems require compatibility with the communication protocol used in that system. OpenTP1 provides such compatibility via particular OpenTP1 products: each product provides compatibility with a different protocol.

Communication via MCF messages rather than RPCs is also possible even in a LAN environment because the product TP1/NET/TCP/IP is available for the TCP/IP protocol.

### 2.3.3 MCF message-exchange configuration using the Extended Presentation facility

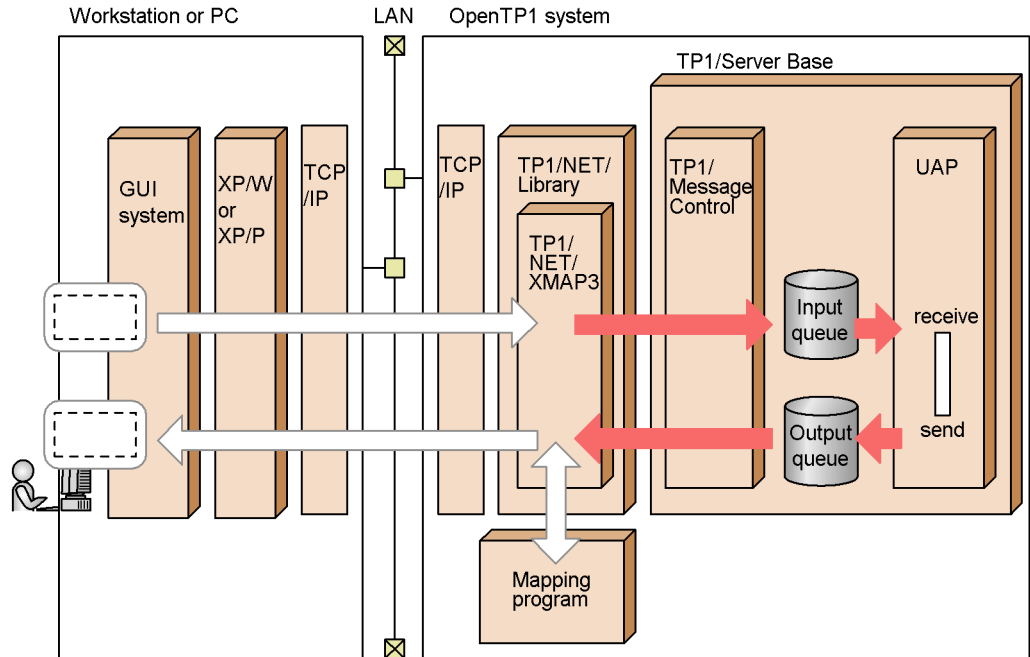
The MCF message-exchange configuration enables communication using the *Extended Presentation facility*. Using this facility, operations such as printing forms and displaying screens on a workstation or PC can be performed by UAP processes in an OpenTP1 system.

The Extended Presentation facility uses TP1/NET/XMAP3 for communications. As a

mandatory requirement, the other machine communicating with the OpenTP1 system node must have the program product XP/W (if a workstation) or XP/P (if a PC) to provide the required GUI-based presentation functionality. There is no need to create a special program for the workstation or PC to serve as the client UAP that requests services in the OpenTP1 system.

Figure 2-4 illustrates the use of the Extended Presentation facility in an MCF message-exchange configuration.

*Figure 2-4: An MCF message-exchange configuration that uses the Extended Presentation facility*



The main features of the Extended Presentation facility are described below.

**(1) Screen data operations using message exchange**

With the Extended Presentation facility, TP1/NET/XMAP3 links with XMAP2/W to perform mapping between physical and logical maps.

When you create a UAP, you do not need to be aware of the screen mapping. Nor do you need to create any special messages to display the data. You can create a message-exchange UAP in the same manner as when handling logical messages in ordinary message exchange.

**(2) Data operations in a top-class GUI environment**

With XP/W or XP/P, you can perform GUI-based operations that make the most of your workstation or PC's excellent presentation functionality. A variety of data operations can be performed from the GUI-based interface.

**(3) System configuration matched to the specific processing configuration**

XP/W and XP/P both use a client/server system. This means that you can build a system conforming to the optimal configuration for application processing between the OpenTP1 system and the workstations or PCs.

For details on TP1/NET/XMAP3 see the manual *OpenTP1 Protocol TP1/NET/XMAP3*. For details on the products XP/W, XP/P, and XMAP2/W, see the manual *XP/W* and the *Extended Mapping Service 2/Workstation XMAP2/W DESCRIPTION/USER'S GUIDE*.

## 2.4 Processing in an MQA message-queuing configuration

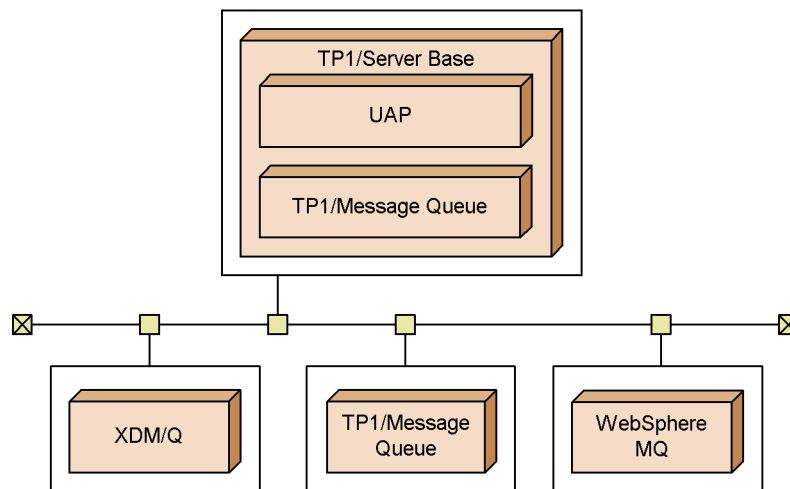
### Queue manager

The queue manager manages MQA message queuing. For message queuing, the queue manager manages communication between applications. The user does not have to worry about matching the communication procedures of UAPs.

The queue manager is required for communication using the MQA message queuing. In the OpenTP1 system, TP1/Message Queue is required as the queue manager. For details of OpenTP1 MQA message queuing, see the *OpenTP1 TP1/Message Queue User's Guide*.

Figure 2-5 shows an example of an OpenTP1 system configuration using MQA message queuing.

*Figure 2-5:* Example of an OpenTP1 system configuration using MQA message queuing



### 2.4.1 Features of MQA message queuing

When MQA message queuing is used, UAPs exchange messages via a queue manager. To put a message, set the destination in the message and catalog it to the queue manager. Then, the queue manager transfers the message to the queue manager of the recipient side. The user of the recipient side can get the message from the queue manager of the recipient side whenever the user decides.

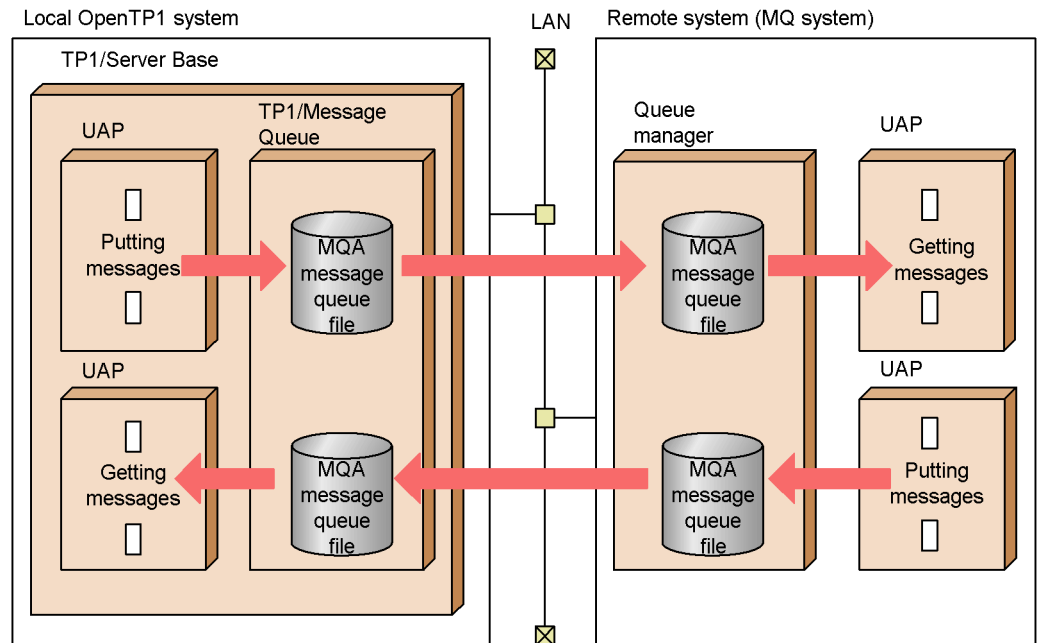
Synchronization with the remote UAP is unnecessary when putting or getting messages. With the MQA message queuing, messages can be used like e-mail because the UAP can handle messages at any time.

Access from UAP to a queue uses MQI (Message Queue Interface). All the queue managers use MQI, so a program the user coded can be used in the system of another queue manager.

The queue that stores the MQA messages from UAPs is an *MQA message queue*. An MQA message queue can contain several queues arranged according to the purpose of the queue manager. For types of queues, see the *OpenTP1 TP1/Message Queue User's Guide*.

Figure 2-6 shows the overview of MQA message queuing.

Figure 2-6: Overview of MQA message queuing



## 2.4.2 Overview of communication using MQA message queuing

This section gives an overview of communication using MQA message queuing.

### (1) Putting messages

A local UAP uses MQPUT to send an MQA message to the queue file in the local system. When MQPUT terminates normally, the message is placed in the local queue file, and the UAP performs its next process. The local queue manager containing the queue file then sends the MQA message to the queue manager in a remote system. The MQA message is placed in the remote queue file for processing by a remote UAP. If you want to communicate interactively, you can define MQPUT to wait for the response from the remote UAP.

## **(2) Getting messages**

Use MQGET to get MQA messages. The timing for receiving MQA messages depends upon whether the message-getting UAP is operating, as described below:

- A local UAP when operating can receive messages from the queue manager by executing MQGET.
- A local UAP that is not operating cannot directly receive messages. The arrival of a message, however, will activate the *trigger facility* in the queue manager. The Trigger facility then sends notification, called a *trigger event*, to the *trigger monitor application*. The trigger monitor application receives the information and decides when to activate the UAP. The user should make sure that the trigger monitor application is active whenever OpenTP1 is active. The user can specify the timing of the activation in the trigger monitor application program.

## **(3) Transaction processing**

OpenTP1 can perform MQA message queuing as a transaction. After the UAP updates data, sends the MQA message to the queue manager, and the transaction is committed, the MQA message is placed in the queue file. If the transaction is rolled back, the MQA message in the queue file can be invalidated. For details of transaction, see *3.1 Transaction Control*.

### **2.4.3 Notes on use of the MQA message queuing**

MQA message queuing enables a UAP to get messages at any time, even though the UAP is not always active. Interactive communication is also available with the MQA message queuing, but the performance may be lower than using RPC.

---

## 2.5 Other Hitachi software products usable with OpenTP1

---

The following software products can be used with OpenTP1 to provide further system development or system management facilities:

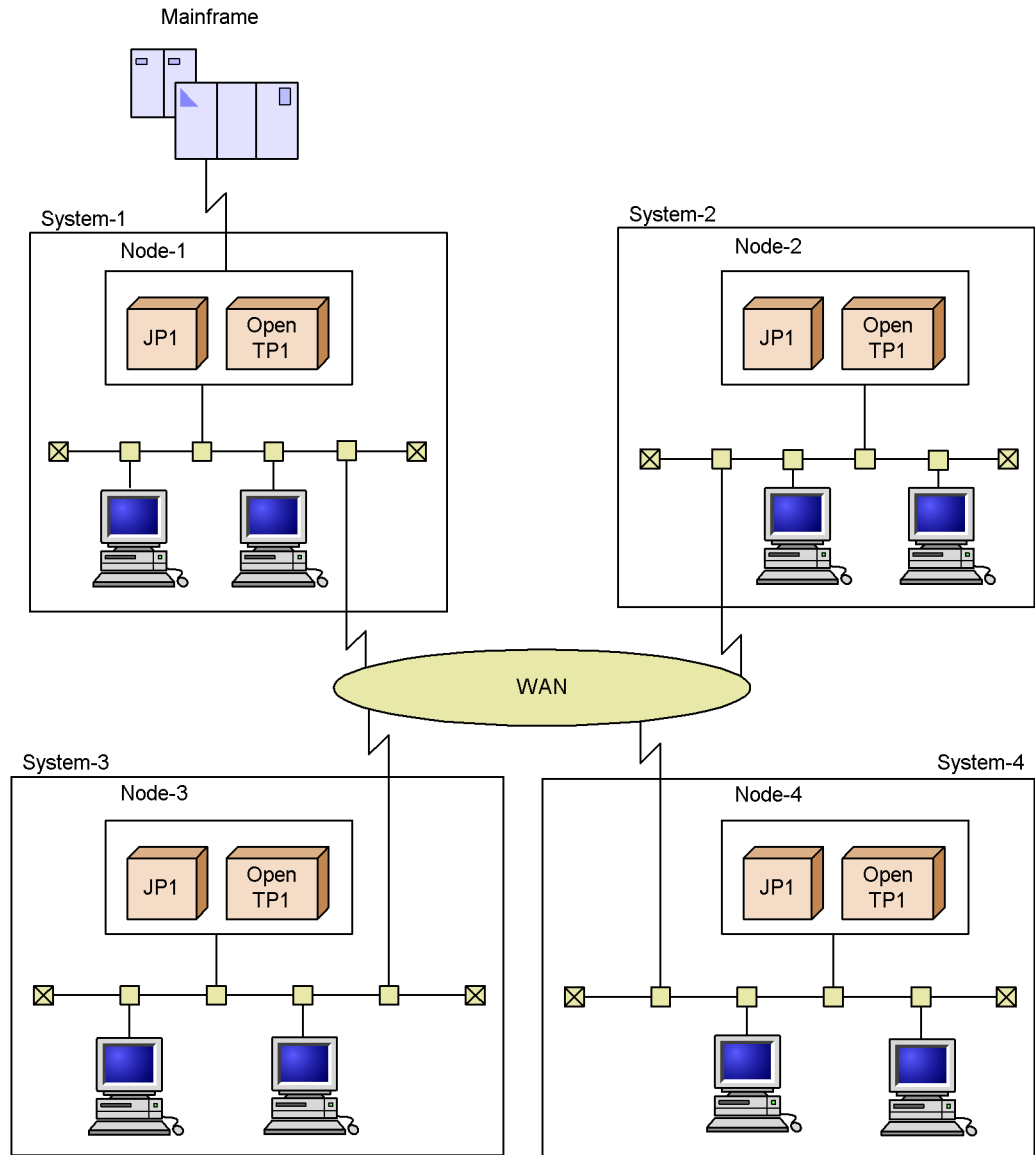
- Job Management Partner 1 Integrated System Operation Management Facility
- SEWB3 Software Engineering Workbench

Using these software products with OpenTP1 allows Job Management Partner 1 to efficiently perform batch job operation, network management, and software distribution in addition to distributed transaction processing by OpenTP1.

### 2.5.1 Job Management Partner 1 Integrated System Operation Management Facilities

The Job Management Partner 1 Integrated System Management Facility comprehensively supports automatic operations and power-saving for the entire system and optimizes Enterprise TCO (Total Cost of Ownership) in a network computing environment. Figure 2-7 shows the OpenTP1 system configuration using Job Management Partner 1.

Figure 2-7: OpenTP1 system configuration using Job Management Partner 1



### Job management

By using Job Management Partner 1 in the OpenTP1 system, batch processing, usually performed by a mainframe computer, can be performed on a UNIX system. Dividing OpenTP1 jobs into online jobs and batch jobs enables more effective use of the system. In addition, by registering OpenTP1 events such as



the start and termination of OpenTP1 in JP1 event service functionality (JP1/System Event Service or JP1/Base), the OpenTP1 system can be automatically run while linked with JP1 job management functionality (JP1/Automatic Operation Monitor or JP1/Automatic Job Management System 2).

#### Distribution and resource management

By using JP1/NETM/DM, programs can be distributed online in a batch.

#### Network management

JP1/Cm2 manages and operates a large, complicated network. By using JP1/Cm2 in the OpenTP1 system, the system can monitor the entire network system from a mainframe computer to a server configured using OpenTP1. In addition, by specifying the log service definition of OpenTP1, JP1/Cm2 enables outputting OpenTP1 terminal output messages to a JP1/Cm2 operation support terminal.

#### System operations using scenario templates

JP1/AJS2 - Scenario Operation is a product for managing execution of JP1/AJS2 jobs and jobnets.

JP1/AJS2 - Scenario Operation lets you create a scenario template, which defines a set of operations, and modify it after system reconfiguration to adapt it to the new system configuration. By using JP1/AJS2 - Scenario Operation, you can execute scenarios appropriate for the operating environment. JP1/AJS2 - Scenario Operation also lets you create and execute a scenario by combining predefined scenario templates provided by OpenTP1. You can use scenario templates to automate system operations.

JP1/AJS2 - Scenario Operation consists of the following components:

- JP1/AJS2 - Scenario Operation Manager

With this component, you can define and save scenario templates, and can manage the execution of scenario templates. You can also register the saved scenario templates as JP1/AJS2 jobnets by connecting to JP1/AJS2 - Manager.

- JP1/AJS2 - Scenario Operation View

With this component, you can operate or monitor JP1/AJS2 - Scenario Operation via a graphical user interface (GUI).

For details on applying scenario templates to system operations by using JP1/AJS2 - Scenario Operation, see *3.10 System operations using scenario templates* or the manual *OpenTP1 Operation*.

For details on using the JP1 product for a specific system operation facility, see the applicable JP1 product documentation.

## 2.5.2 SEWB3 Software Engineering Workbench

The SEWB3 Software Engineering Workbench is a Hitachi system development

support tool that aids programming on workstations. SEWB3 has the following features.

- Integrates and manages information required for program development
- Enables the efficient development of programs by using graphics to express design specifications and program logical structures.

By using SEWB3 in the OpenTP1 system, UAPs and user exit routines can be developed efficiently. In addition, you can use the fourth generation language SEWB3/4GL as well as C, C++, and COBOL85.

For details about the products related to SEWB3, see the manual *SEWB3 General Information*.

---

## 2.6 User application programs in OpenTP1 systems

---

This section describes UAPs (user application programs) used in OpenTP1 systems.

### 2.6.1 User application programs and types of processing

OpenTP1 uses the following types of UAPs:

#### (1) UAPs for client

- Service-using program (SUP)

Requests services from service-providing programs. SUPs are used exclusively as client UAPs and cannot provide any services to another UAP. SUPs are used in the TP1/Server Base product.

- Client user program (CUP)

A UAP exclusively used by clients when they use the OpenTP1 client software (TP1/Client/W or TP1/Client/P).

#### (2) UAPs for server

- Message-handling program (MHP)

Used in a message-exchange configuration. An MHP works with MCF and receives and processes a message sent from a non-OpenTP1 system.

#### (3) UAP for offline work

- UAP that handles offline work

Uses an OpenTP1 function while offline (e.g., initialize a user file for use as a DAM file). This type of UAP performs user-determined processing.

#### (4) UAP used by the OpenTP1 client facility (TP1/Client)

- Client user program (CUP)

A client-specific UAP that uses the OpenTP1 client facility (TP1/Client/W or TP1/Client/P).

With TP1/Client/J, you can create Java applets, Java applications, and Java servlets that request SPP services.

### 2.6.2 Overview of user application programs

The following sections provide overviews of the following types of UAPs:

- service-using programs
- service-providing programs

- message-handling programs
- UAPs that handle offline work

For more details of UAPs used in OpenTP1 systems, see the *OpenTP1 Programming Guide*.

**(1) SUP: service-using program**

A SUP is an OpenTP1 client UAP that starts the application processing in client/server communications. A SUP can control the start and finish of a transaction, and request a service from a server UAP. A SUP is used as a client only and cannot also function as a server UAP to provide a service to another UAP.

A SUP can be started at a user-determined time: at the same time as OpenTP1 starts, or online with the OpenTP1 command `dcsvstart`. In a user service configuration definition, you can specify whether a SUP is to start at the same time as OpenTP1.

OpenTP1 does not control the termination of a SUP so the programmer must code the SUP so that it terminates at the end of application processing.

**(2) SPP: service-providing program**

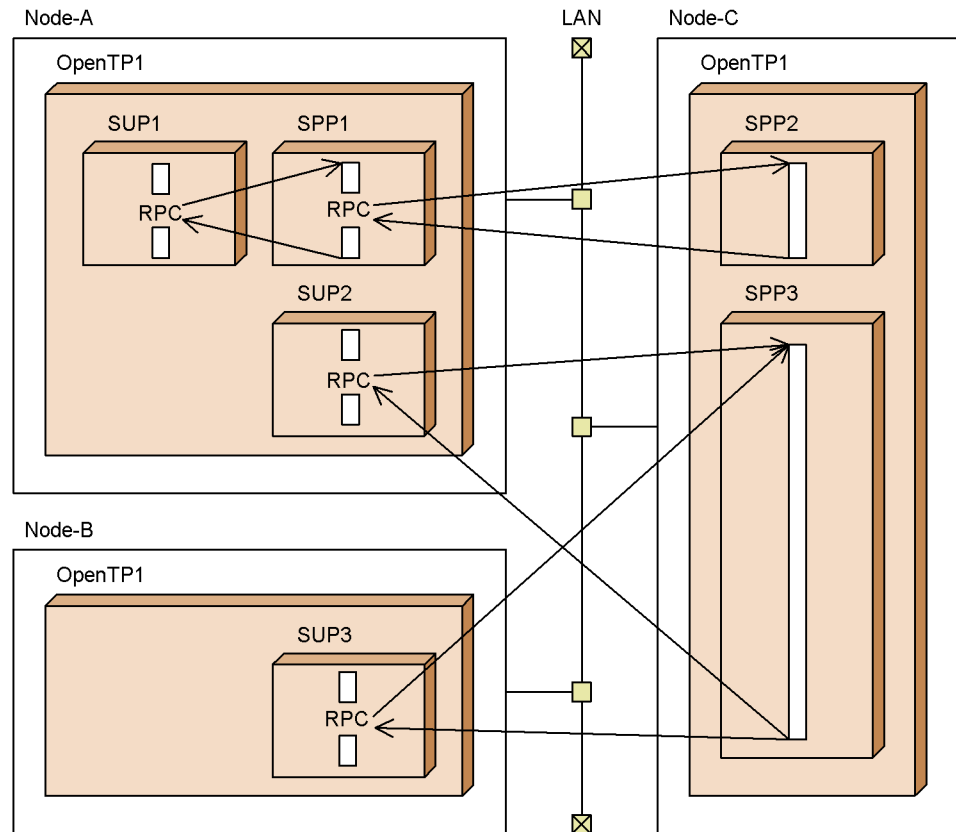
An SPP provides a service as a user server in client/server communications. An SPP processes requests for a service: the requests are sent by client programs such as a SUP or CUP. An SPP can also start a transaction or send messages.

An SPP can be started at a user-determined time: either started online with the OpenTP1 command `dcsvstart`, or started at the same time that OpenTP1 starts. In the latter case, when OpenTP1 starts, the SPP is made ready to receive requests. In a user service configuration definition, you can specify whether an SPP is to start at the same time as OpenTP1.

SPPs can be created for each work process, and provide required processing as services. Required services can be added as a system expands. This makes such services easy to maintain and easy to expand.

An SPP can request a service from another SPP, so processing can be nested. Figure 2-8 illustrates such a situation and the relation between SUPs and SPPs.

Figure 2-8: SUPs and SPPs in OpenTP1 systems

**(a) SPP structure**

When writing an SPP in C, C++, or COBOL, the programmer codes SPP services as functions. An individual function is called a *service function* in C or a *service program* in COBOL. The function that organizes the component functions is called the *main function* in C or the *main program* in COBOL.

**(b) Compilation and linkage**

To create an SPP executable file, the SPP source program is compiled and linked. At linkage, the stubs that define the entry point for each service function are linked. A *stub* is a code module that provides an interface for RPCs between a client and server.

A stub is unnecessary when service functions are created in a UAP shared library<sup>#</sup> and loaded dynamically. For details about dynamic loading, see 3.8 *Dynamic loading of service functions*.

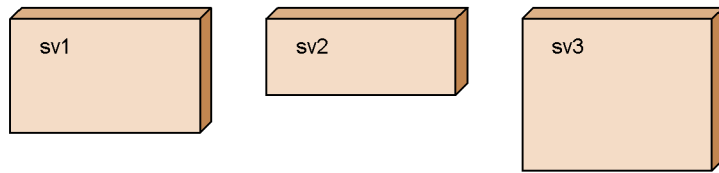
#

This involves compiling the UAP source files and linking the created UAP object files into a shared library.

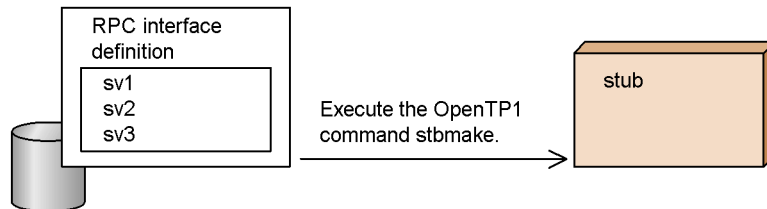
The two figures below show the SPP structure when a stub is used and when dynamic loading of service functions is used. Figure 2-9 shows the structure of an SPP (when using a stub). Figure 2-10 shows the structure of an SPP (when using dynamic loading of service functions). For details about creating a UAP, see the *OpenTP1 Programming Guide*.

Figure 2-9: SPP structure (when using a stub)

1. Create the services, which are to be provided to a client UAP, as service functions.



2. Create the stub which defines the entry points for all services.



3. To produce an executable SPP file, create the main function then compile and link the main function, service functions, and stub.

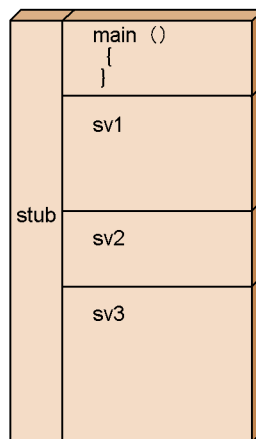
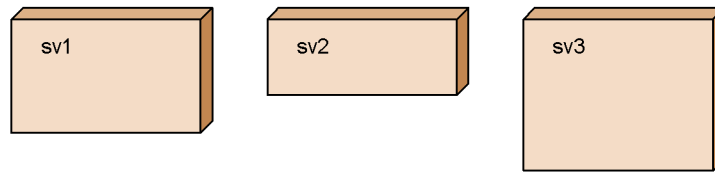
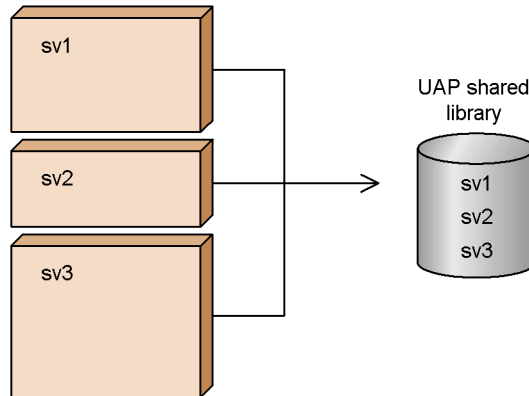


Figure 2-10: SPP structure (when using dynamic loading of service functions)

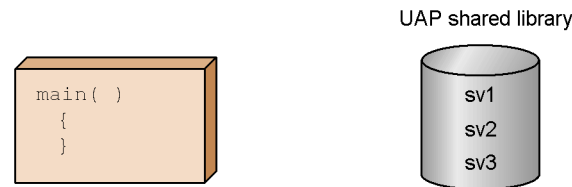
1. Create the services, which are to be provided to a client UAP, as service functions.



2. Compile and link the service functions into a UAP shared library.



3. Create and compile the main function to create an SPP executable file. When the SPP is executed, the service functions are loaded and executed from the UAP shared library created at step 2.



**(c) Relationship between SPPs and RPCs**

In a user service definition, you can specify an SPP executable file as a *service-group name*, and specify each service function as a *service name*. Such service names correspond to entry points specified in RPC interface definitions. When writing code for a client UAP to request a service, the programmer must specify the SPP service group name and service name in the RPC function parameters.



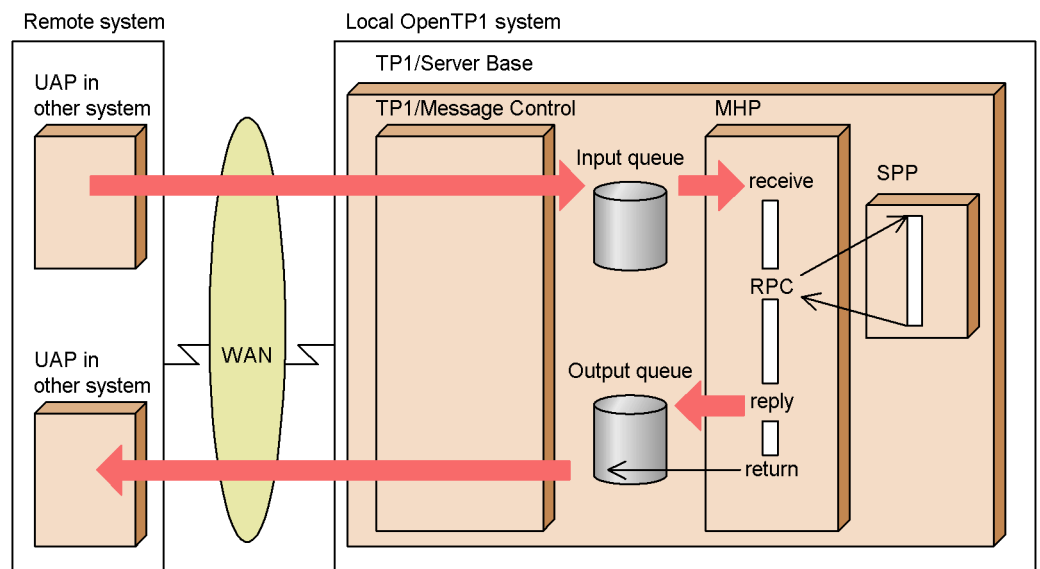
### (3) MHP: message-handling program

An MHP is used in message-exchange configurations and is used exclusively for processing messages. An MHP receives messages from other systems and processes the messages. An MHP can use the various MCF services that, for example, enable the sending or receiving of messages. A node on which an MHP runs requires TP1/Message Control. An MHP can also request a service from an SPP.

An MHP starts application processing when a message is received. In the user service configuration definition, you can specify whether an MHP is to start at the same time as OpenTP1. If an MHP starts when OpenTP1 starts, messages can be received from the start of online processing. If an MHP does not start when OpenTP1 starts, the MHP can be started at a user-determined time using the OpenTP1 command `dcsvstart`.

Figure 2-11 shows how MHP functions in an OpenTP1 system.

Figure 2-11: MHP in an OpenTP1 system



#### (a) MHP structure

The unit or part of a UAP that processes a message during message-exchange processing is called an *MCF application*. The programmer codes the processing of an MHP that corresponds to an MCF application as a function. An individual function is called a *service function* in C or a *service program* in COBOL. The function that organizes the created service functions is called the *main function* in C or the *main program* in COBOL.

**(b) Compilation and linkage**

To create an MHP executable file, the MHP source program is compiled and linked. At linkage, the stubs that define the entry point for each service function are linked. A stub is created after creating the RPC interface definition and then executing the `stbmake` command.

Note that stubs are not needed if you create all service functions as a shared UAP library<sup>#</sup> and then perform dynamic loading of service functions. For details about dynamic loading of service functions, see *3.8 Dynamic loading of service functions*.

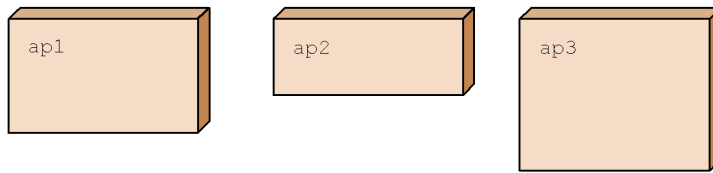
#

A shared UAP library is a shared library that contains UAPs created by compiling the UAPs' source files and then linking the resulting UAP object files.

The two figures below show the MHP structure when a stub is used and when dynamic loading of service functions is performed. For details on writing MHPs, see the *OpenTPI Programming Guide*.

Figure 2-12: MHP structure (when using a stub)

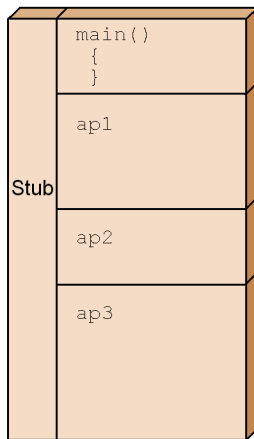
1. Create as service functions the services that are to be provided to client UAPs.



2. Create the stub that defines the entry points for all service functions.

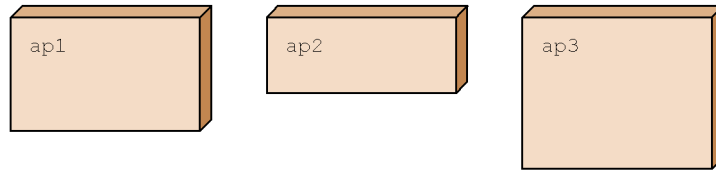


3. To obtain an executable MHP file, create the main function, and then compile and link the main function, service functions, and stub.

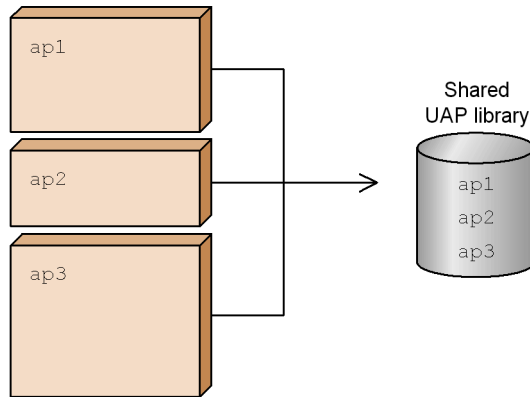


*Figure 2-13: MHP structure (when performing dynamic loading of service functions)*

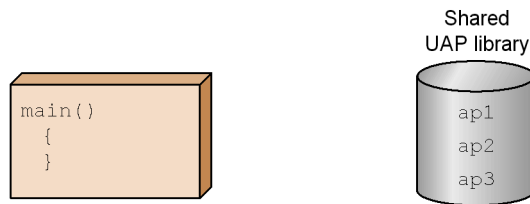
1. Create as service functions the services that are to be provided to client UAPs.



2. Create a shared UAP library of service functions.



3. To obtain an executable MHP file, create the main function and then compile it. When the MHP is started, the service functions are obtained from the shared UAP library that was created in step 2, and then executed.



**(4) Relationship between an MHP service name and an application**

In a user service definition, you can specify an MHP executable file as a *service-group name*, and specify each service function as a *service name*. Such service names correspond to entry points specified in RPC interface definitions. The service-group name and service name are specified in the RPC function parameters.

A service name specified in a user service definition corresponds to an application name in an MCF application definition. OpenTP1 uses this application name as the basis for processing.

#### (a) Types of MHP applications

MHP services (applications) can be classified according to the kind of message processing, which is specified in the application attribute part of the MCF application definition. The kinds of MHPs are:

ans

An MHP that returns an answer (reply) to the other system.

noans

An MHP that does not return an answer (reply) to the other system.

cont

An MHP that continues sending or receiving messages with the other system.

#### (5) UAP that handles offline work

Programmers can write a UAP that can access a DAM file offline. In a UAP that handles offline work, the programmer must ensure that the DAM file is initialized for use as a DAM file before OpenTP1 starts. A UAP that handles offline work can use the OpenTP1 facilities for initializing DAM files but cannot use the OpenTP1 facilities for RPCs or message-processing.

A user controls the starting and terminating of a UAP that handles offline work, and starts the UAP from the UNIX prompt.

### 2.6.3 Cooperation of user processes with SPPs and MHPs

To share and thereby reduce processing loads, OpenTP1 can distribute the processing of a service group across multiple processors. This feature is called the *Multiserver facility* and is described in more detail in *3.4.3 Process control and the Multiserver facility*. The Multiserver facility equalizes loads on various processors by passing a service request to any waiting process that can process the same service group.

In OpenTP1, you can define the maximum number of processes that can be executed for each service group.

### 2.6.4 UAP testing and debugging facilities

Before using a created UAP in an OpenTP1 system, you can test whether the UAP processing works. OpenTP1 provides an Offline Tester, Online Tester, and MCF Online Tester. For details, see the *OpenTP1 Tester and UAP Trace User's Guide*.

#### (1) Offline Tester

The Offline Tester is used for offline testing of UAPs that will be used for online

processing. The Offline Tester is used before a UAP is used in an OpenTP1 system.

The Offline Tester can test the behavior of an SPP or MHP. Workstations on which the Offline Tester is executed require the product TP1/Offline Tester.

**(2) Online Tester**

The Online Tester works with OpenTP1 to test UAPs in an online environment. Before using the UAP in actual processing, you can test its behavior with OpenTP1 services. The Online Tester can test the behavior of an SUP or SPP. You can also test an MHP as an SPP. Workstations on which the Online Tester is executed require the product TP1/Online Tester.

**(3) MCF Online Tester**

The MCF Online Tester works with TP1/Message Control to test MHPs in an online environment. As with the Online Tester, the behavior of MHPs can be tested while using OpenTP1 services online.

Workstations on which the MCF Online Tester is executed require the product TP1/Message Control/Tester.

## 2.7 Processing in an Internet-based configuration

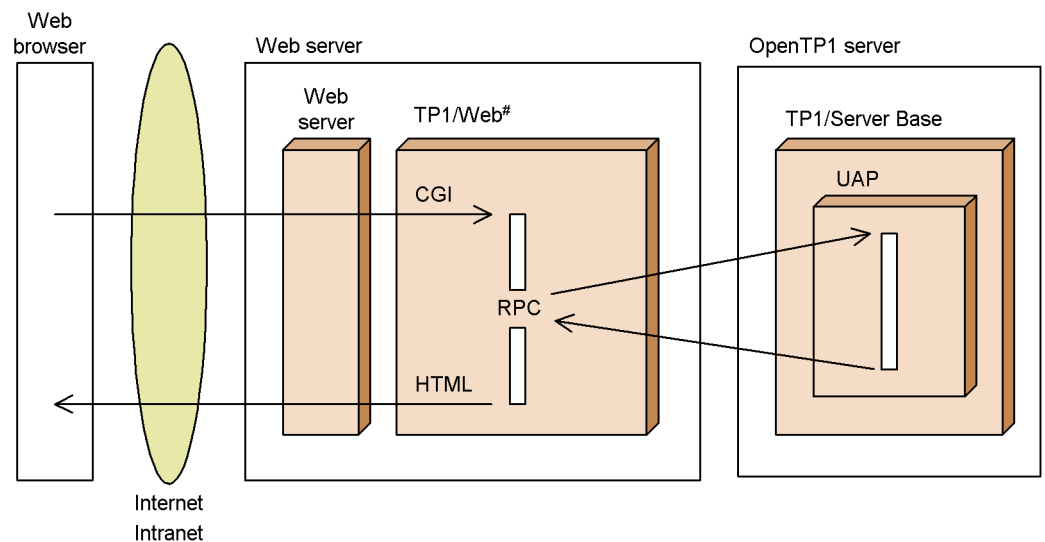
TP1/Web converts Internet-based information exchange between a browser and a Web server into OpenTP1 remote procedure calls, and executes UAPs on an OpenTP1 server. Using this functionality, you can build a processing system that uses a Web browser as a terminal.

When linked with OpenTP1, TP1/Web can be used in the following types of processing systems:

- Online marketing and shopping
- Internet-based travel or event booking system, or hotel room booking system
- Corporate online systems that can be accessed via the Internet from domestic or overseas company sites or from another location during a business trip

TP1/Web can be used not only when connected to the Internet but also via an enterprise LAN. Figure 2-14 shows a system configuration using TP1/Web.

Figure 2-14: Configuration with TP1/Web



Legend:

CGI: Common Gateway Interface

HTML: Hyper Text Markup Language

#: TP1/Client is a prerequisite for TP1/Web





## Chapter

---

# 3. Functions

---

This chapter describes the service functions provided by OpenTP1.

- 3.1 Transaction Control
- 3.2 Processing in an OpenTP1 client/server configuration
- 3.3 Message Control
- 3.4 Scheduling
- 3.5 OpenTP1 client facility (TP1/Client)
- 3.6 Client/server communications using OSI TP
- 3.7 Remote API facility
- 3.8 Dynamic loading of service functions
- 3.9 Additional Features
- 3.10 System operations using scenario templates
- 3.11 System monitoring using audit logs

---

## 3.1 Transaction Control

---

The application processing programs used in a distributed computer environment need to be highly reliable. OpenTP1 delivers a variety of transaction control functions for such programs. Using these functions when running OpenTP1 reduces the overhead of dealing with the problems peculiar to a distributed system.

### 3.1.1 Distributed transactions

In OpenTP1 systems a transaction is a logical unit of work in which a group of related task steps is handled as a single unit. When the transaction is successful, a *commit operation* is performed so that all the steps that make up a transaction have their desired effects. When the transaction is not successful, a *rollback operation* is performed so that none of the steps that make up a transaction have an effect. By such operations OpenTP1 ensures that at the end of a transaction the resources are in a *consistent* state: either all the steps in a transaction have the desired effects, or none of the steps in a transaction have an effect.

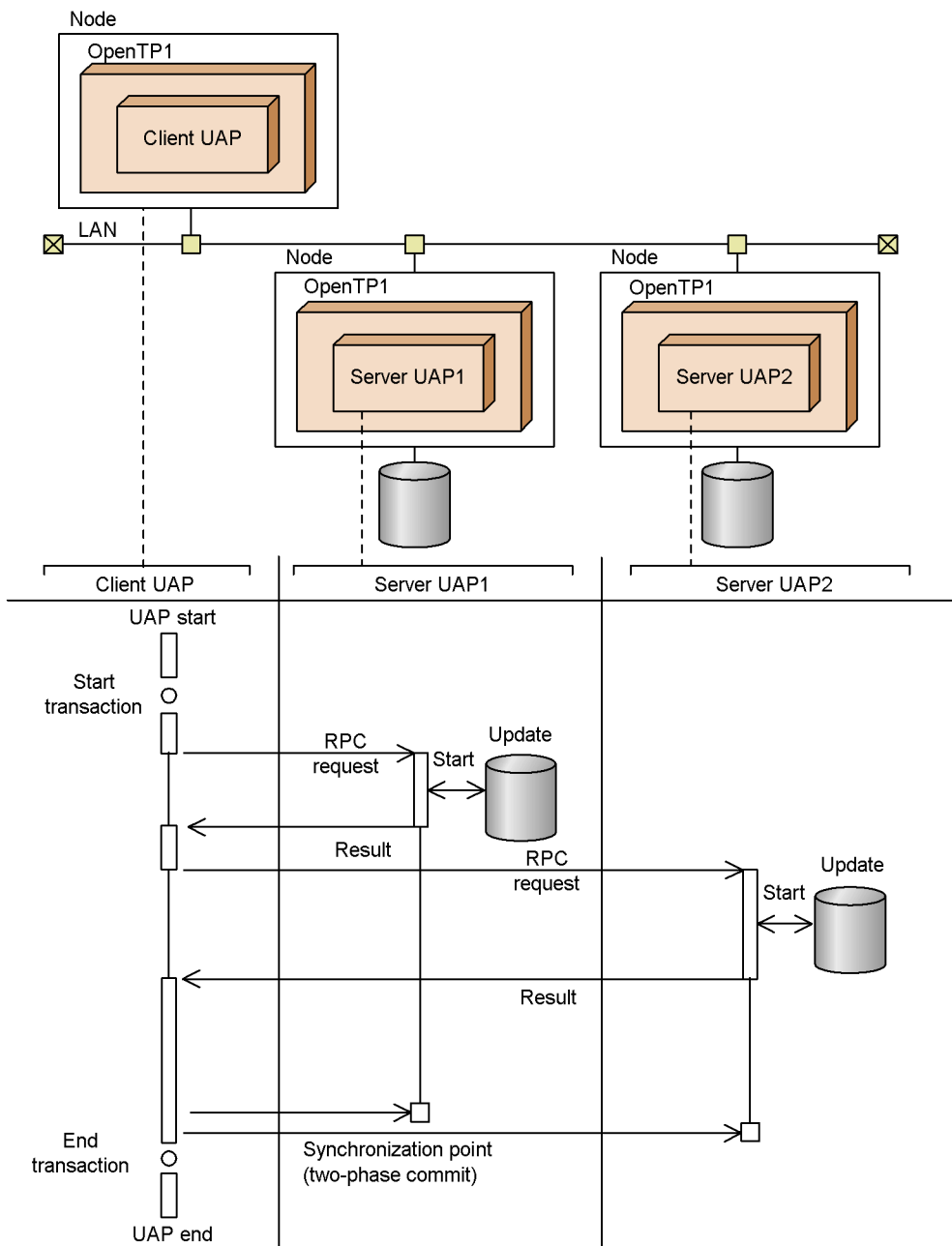
Previous online transaction processing systems were either controlled by a single large-scale computer (in which case the consistency of a transaction could be guaranteed only within that one computer) or by a specialized DBMS (in which case only specialized database operations could be treated as transactions). OpenTP1, however, provides general online transaction processing in which parts of a transaction are processed at different locations and on different machines. This type of transaction processing is called *distributed transaction processing*.

In online distributed transaction processing, OpenTP1 manages distributed resources so that they are maintained in a consistent state. Transactions can be executed concurrently without adversely affecting each other: for example, OpenTP1 ensures that more than one transaction never updates the same resource at the same time.

OpenTP1 maintains the consistency of resources even when some mechanical or communication failure prevents immediate updating of resources. By automatically collecting information relating to transactions and the resources, OpenTP1 can ensure that failure recovery is possible even in the face of significant mechanical or communication failure.

Figure 3-1 shows a distributed transaction.

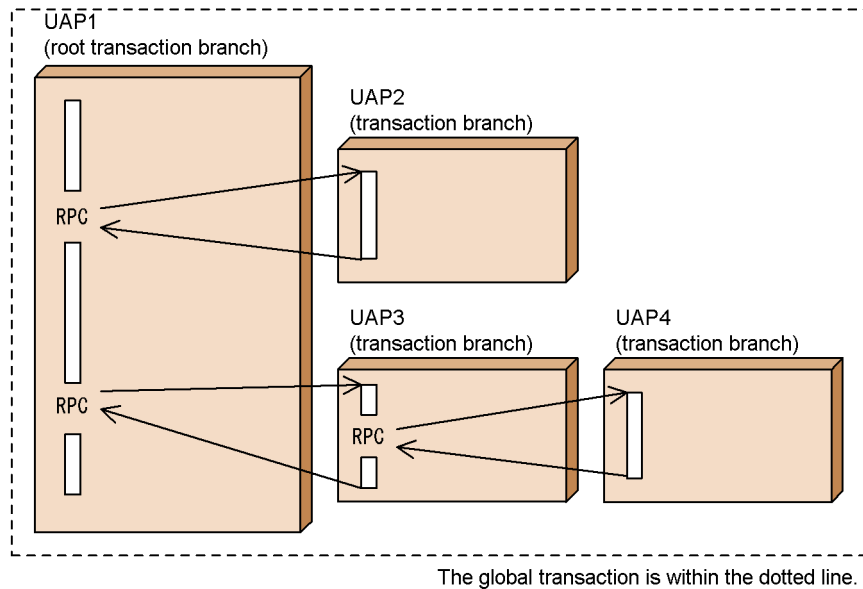
Figure 3-1: A distributed transaction



### 3.1.2 RPCs, transaction branches, and global transactions

A UAP process for processing a transaction is called a *transaction branch*. You can use an RPC to process a transaction that consists of multiple UAP processes. A set of transaction branches that consists of multiple UAP processes is called a *global transaction*. The process that starts the transaction is called the *root transaction branch*. Figure 3-2 shows an example of a global transaction.

Figure 3-2: A global transaction and transaction branches



### 3.1.3 Commit and rollback operations

A transaction ends with success (with a commit operation performed on the transaction) or failure (with a rollback operation performed on the transaction). The decision of whether a transaction is to end in success or failure (i.e., whether to perform a commit or rollback operation) is called a *transaction determination*. OpenTP1 performs a transaction determination at a *synchronization point*. This point is at the boundary of two transactions: at the end of one and at the beginning of the next. Resources are updated at a synchronization point.

In a *commit operation on a transaction*, the transaction ends successfully with affected resources updated consistently. During the commit operation, the affected resources for all the transaction branches in a global transaction are updated.

In a *rollback operation on a transaction*, the transaction ends in failure with those resources that should have been updated by the transaction returned to the same status as at the beginning of the transaction. If a UAP detects data inconsistency or a failure

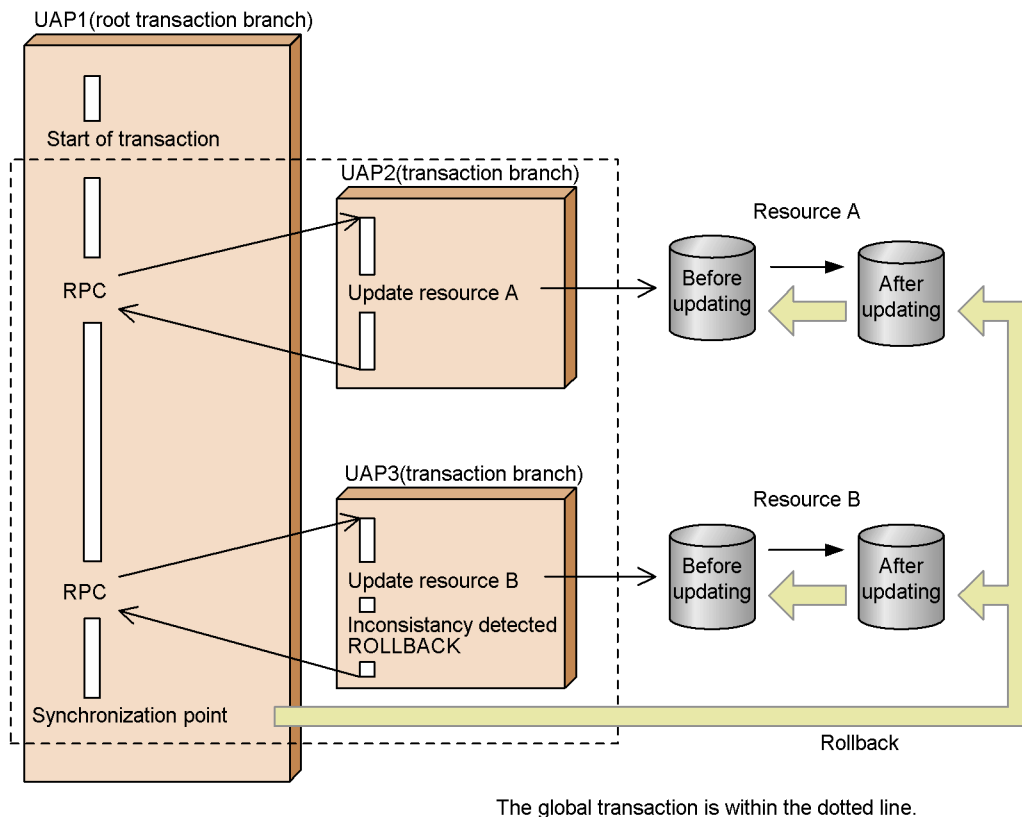
before a synchronization point, rollback functions (e.g., `dc_trn_unchained_rollback()` or `dc_trn_chained_rollback()`) can be issued to rollback the transaction and preserve the integrity of the data.

OpenTP1 can log the reason why the transaction rolled back. The operand `trn_rollback_information_put` in the transaction service definition allows you to specify whether or not to log the causes of transaction rollback.

OpenTP1 monitors the times for each transaction branch. If a transaction branch does not end in the allotted time, OpenTP1 rolls back the transaction. For details of time monitoring, see the manual *OpenTP1 System Definition*.

Figure 3-3 shows a rollback operation (indicated by the gray line) on a transaction.

Figure 3-3: Rollback operation on a transaction



### 3.1.4 Two-phase commit

#### (1) Two-phase commit

When multiple resources are to be updated, at a synchronization point OpenTP1

determines whether or not to update the resources in a procedure called a *two-phase commit*. The two-phase commit separates a commit operation into two phases:

- prepare processing
- commit processing

The *prepare processing* is preparing to update resources. The *commit processing* is actually updating the resources. Separating the commit operation into two phases enables multiple resources to be updated without contradiction.

As illustrated in Figure 3-4, in the first phase, at a synchronization point the root transaction branch requests all the transaction branches in the global transaction to prepare for a commit operation. Each transaction branch then notifies the root transaction branch whether it completed preparation for the commit operation or whether it performed a rollback operation.

In the second phase, at the synchronization point OpenTP1 performs a transaction determination to decide whether a commit operation or a rollback operation should be executed on the global transaction:

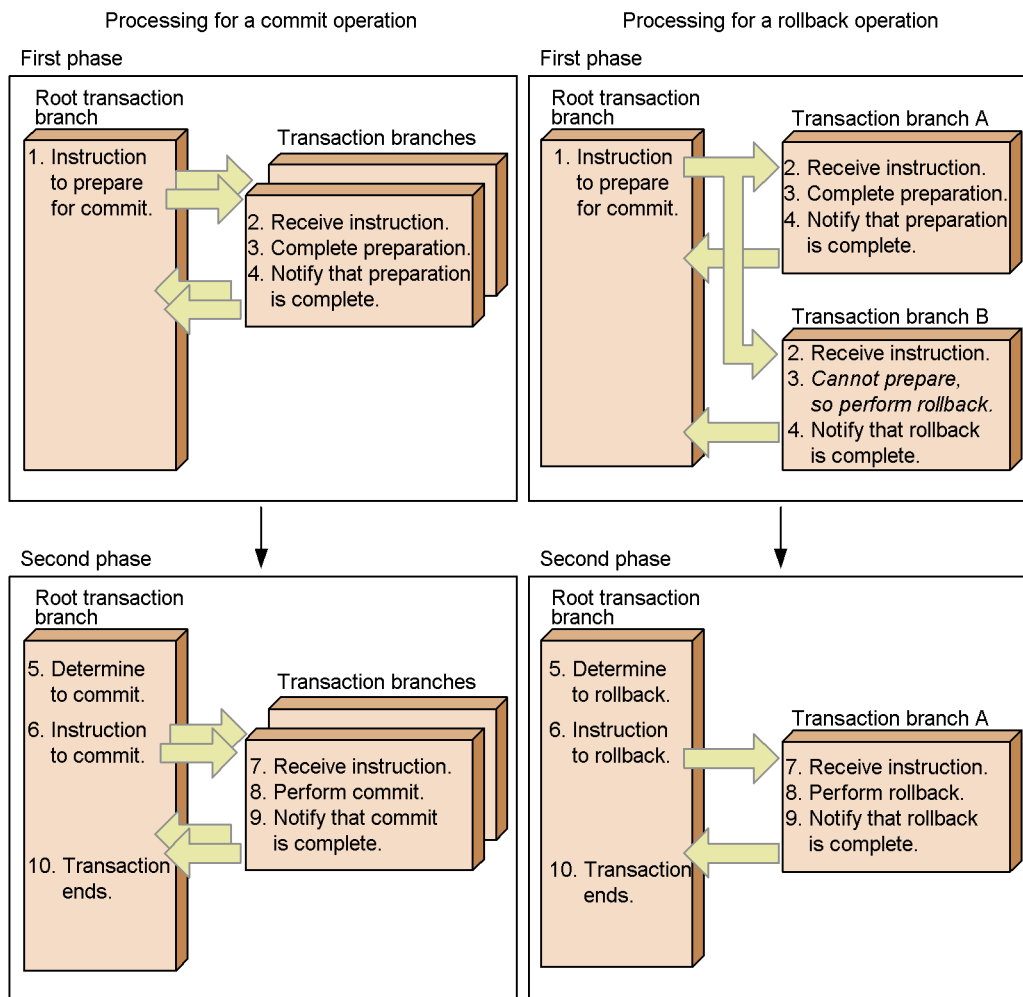
- If all the transaction branches completed their preparations for the commit operation, the root transaction branch orders a commit operation on the entire global transaction.
- If one or more transaction branches performed rollback operations during the first phase, the root transaction branch orders a rollback operation on the entire global transaction.

All the transaction branches in the global transaction follow the decision of the root transaction branch.

If a system failure occurs after the transaction determination, the history information for the transaction during phase 2 processing is collected into a system journal: the TJ transaction journal.

Figure 3-4 shows the processing for a two-phase commit operation.

Figure 3-4: Processing in a two-phase commit



**(2) Heuristic decisions**

If a communication error occurs after the root transaction branch instructs the transaction branches to prepare for a commit operation, the root transaction branch's decision of whether to perform a commit or rollback operation might not be sent to the transaction branches. This is called a *heuristic hazard*.

If a heuristic hazard occurs, the OpenTP1 administrator can use a command (e.g., `trncmt`, `trnrbk`, or `trnfgt`) to forcibly apply the results of a transaction determination to each transaction branch. In a *heuristic decision*, the results of a transaction determination are applied to an individual transaction branch regardless of

the decision of the root transaction branch. In a *heuristic commit operation* the commit operation is executed for an individual transaction branch. In a *heuristic rollback operation* the rollback operation is executed for an individual transaction branch.

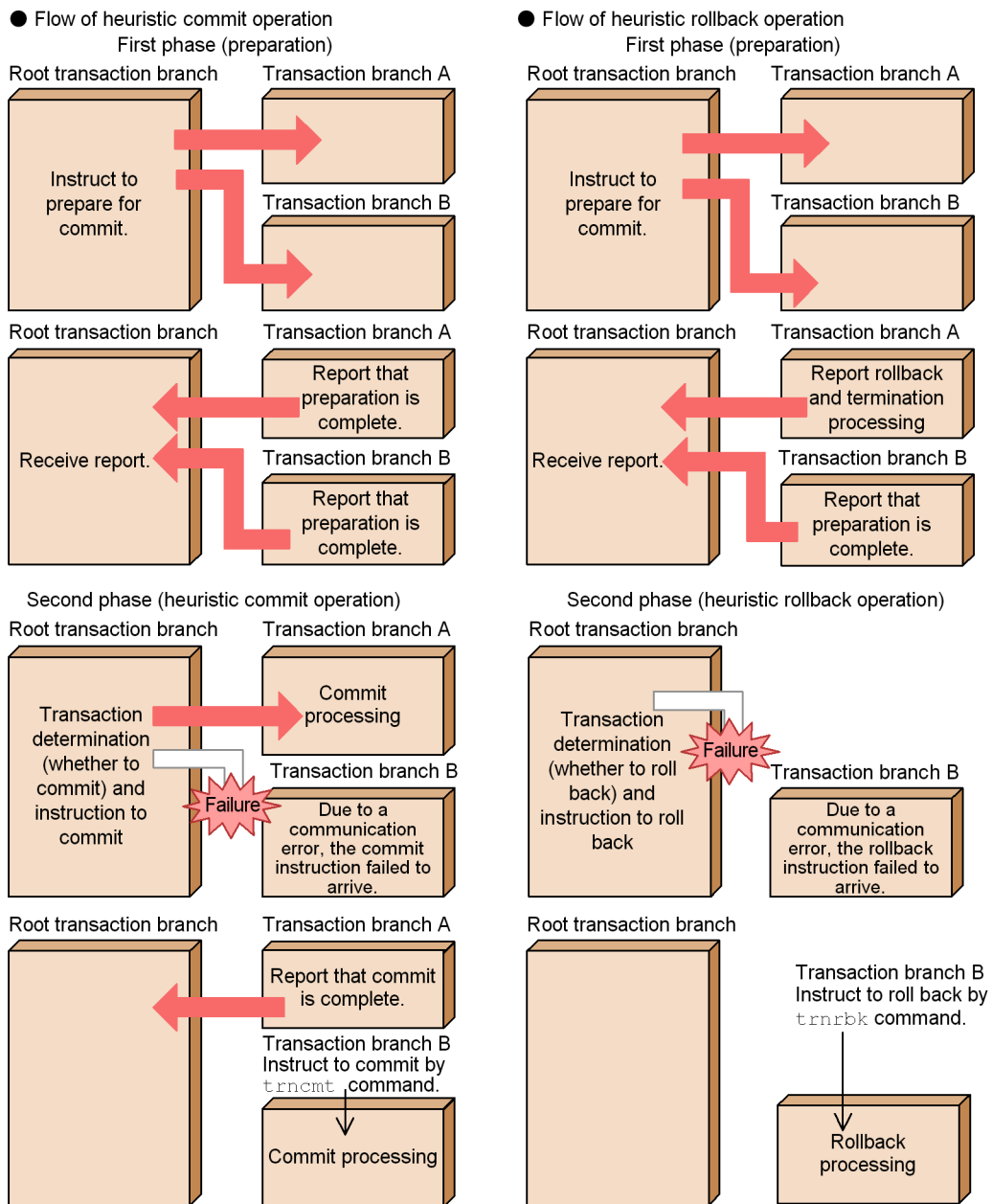
The OpenTP1 administrator must avoid conflicts between the transaction determination of the root transaction branch and the individual determinations at each transaction branch. To avoid such conflict the user should:

- use the status display command `trnls` to check whether the transaction determination at the root transaction branch resulted in a decision to perform a commit operation or a rollback operation
- force the transaction branch to perform the same operation (either commit or rollback) that was performed at the root transaction branch.

The user is responsible for any forced determination of a transaction caused by a command such as `trncmt`, `trnrbk`, or `trnfgt`. If a user uses a command to perform a forced determination at a transaction branch, OpenTP1 does not assure the consistency of resources. A conflict between the transaction determination of the root transaction branch and the action at a transaction branch is called a *heuristic mix*. Users must take care to avoid causing heuristic mixes when using a command to perform a forced determination. Figure 3-5 shows the processing for a heuristic decision.



Figure 3-5: Processing in a heuristic decision



### 3.1.5 Transactions and UAPs

In the user service definition, a program developer should specify the UAPs to be

included in a global transaction. Note that the user service definition must be specified such that MHPs are included in global transactions. A UAP defined to be included in a global transaction is called a UAP that has the *transaction attribute*.

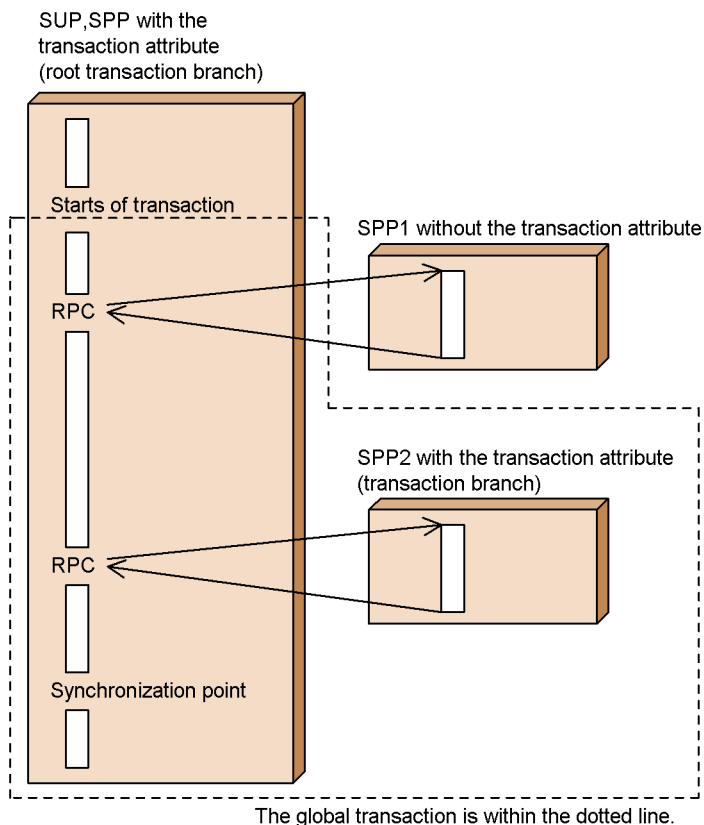
**(1) Transactions and SUPs and SPPs**

For SUPs and SPPs, a function (e.g., `dc_trn_begin()`) declares the start of a transaction. Another function (e.g., `dc_trn_chained_commit()` or `dc_trn_chained_rollback()`) sets a synchronization point. Whenever a function that sets a synchronization point is issued, OpenTP1 performs a transaction determination for the global transaction of the SUP or SPP. Only an SUP or SPP that has the root transaction branch can issue a function to perform a transaction determination.

Only an SUP or SPP with the transaction attribute can issue a function that starts a transaction or sets a synchronization point. You can reduce the overhead required for transaction processing by using the user server definition to specify that the UAPs that access resources are included in a global transaction and that no other UAPs are included in the global transaction. To access resources, an SUP or SPP must be defined in the user service definition as being included in a global transaction.

Figure 3-6 shows a global transaction for an SUP or SPP.

Figure 3-6: Global transaction for a SUP or SPP

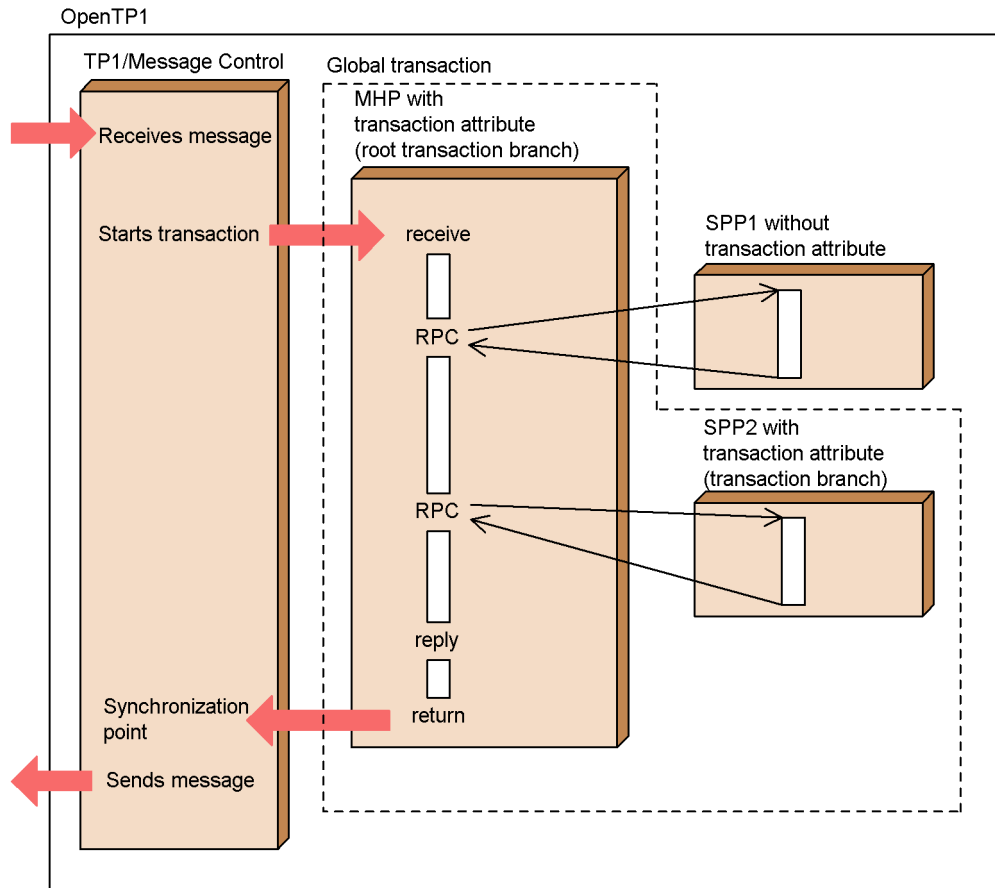


**(2) Transactions and MHPs**

When an MHP service is called, transaction processing starts. An MHP cannot issue a function (e.g., `dc_trn_begin()`) that starts a transaction. In cases where the MHP requests a service from an SPP, the MHP becomes the root transaction branch. Also, the SPP from which a service is requested cannot start a transaction.

Figure 3-7 shows a global transaction for an MHP.

Figure 3-7: Global transaction for an MHP



### 3.1.6 Transaction control based on the TX interface

OpenTP1 supports transaction control functions (*TX functions*) that conform to the DTP model defined in X/Open. For details about using these functions, see the *OpenTP1 Programming Guide*.

### 3.1.7 Transaction control based on the XA resource service

The XA resource service coordinates transactions between OpenTP1 and the following resource, using a two-phase commit:

- An application server running on the J2EE platform
- A .NET Framework application (application running on .NET Framework)

The functionality that links OpenTP1 with a .NET Framework application is known as

*MSDTC linkage.* MSDTC linkage enables transactions to be linked between an OpenTP1 resource and an MSDTC resource. To use MSDTC linkage, *Y* must be specified in the `xar_msdtc_use` operand in the XA resource service definition.

This section provides an outline of transaction control based on the XA resource service. For details about operation, see the description of the XA resource service in the manual *OpenTP1 Operation*. For details about trace information related to the XA resource service, see 5.3.6(5) *XAR performance verification trace* and 5.3.6(9) *XAR event trace*.

In this manual, where there is no functional difference between an application server running on J2EE and a .NET Framework application, the term *application server linked by the XA resource service* is used.

### **(1) Overview of the XA resource service**

The main purpose of the XA resource service is to manage the status of transaction identifiers (XIDs) passed by an application server linked by the XA resource service, and to map them with OpenTP1's XIDs. When MSDTC linkage is used, the XA resource service also manages transaction recovery information (RI). *RI* is information created by MSDTC during transaction determination so that the transaction can be recovered if an error occurs in the processing. The transaction status is recorded in the OpenTP1 file system as required.

Note:

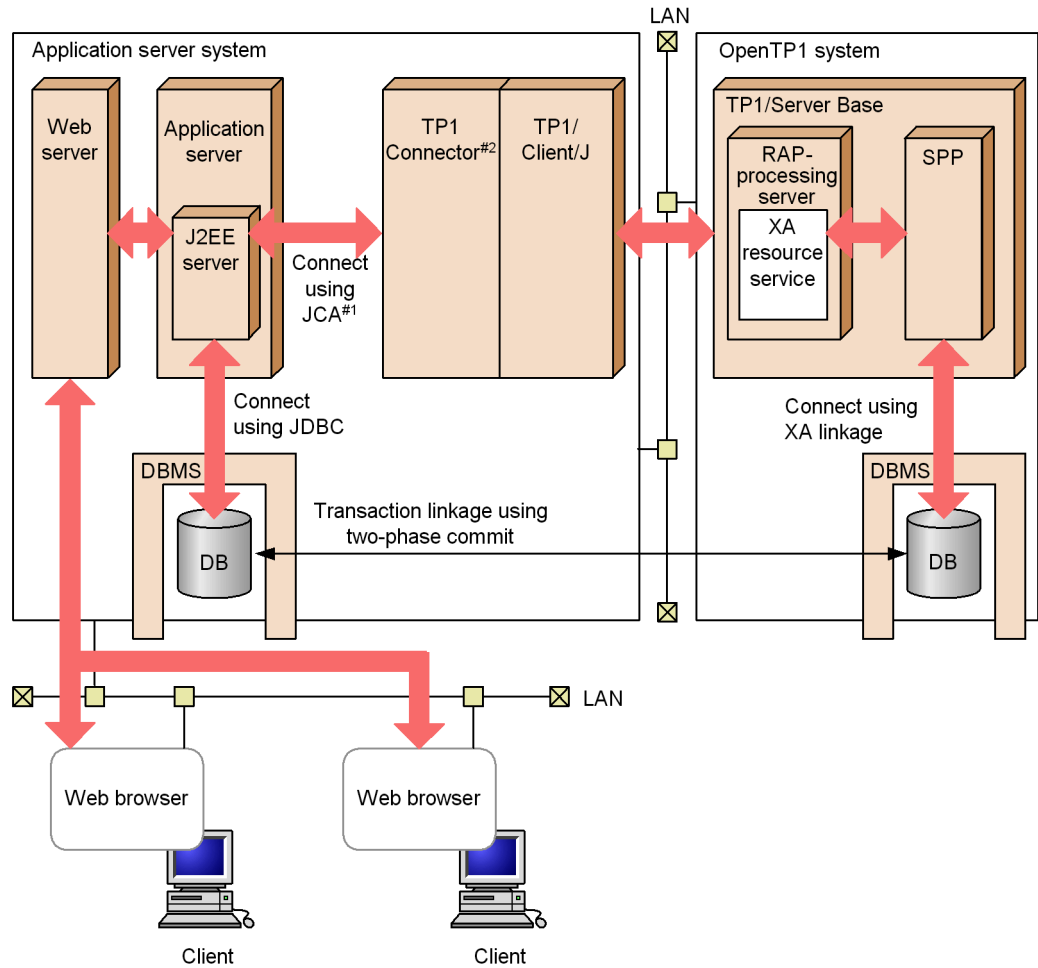
The *XA* in *XA resource service* has a different meaning from the *XA* of the *XA linkage function* used by OpenTP1 to instruct the DBMS to perform a transaction determination.

#### **(a) Linkage with a J2EE application server**

An application server that runs on J2EE controls transactions via the uCosminexus TP1 Connector or Cosminexus TP1 Connector, which complies with J2EE Connector Architecture. OpenTP1 receives transaction requests from the J2EE application server on a RAP-processing server, and processes transactions using the XA resource service.

Figure 3-8 shows the flow of transaction control when OpenTP1 is linked with an application server that runs on J2EE.

Figure 3-8: Transaction control when linked with a J2EE application server



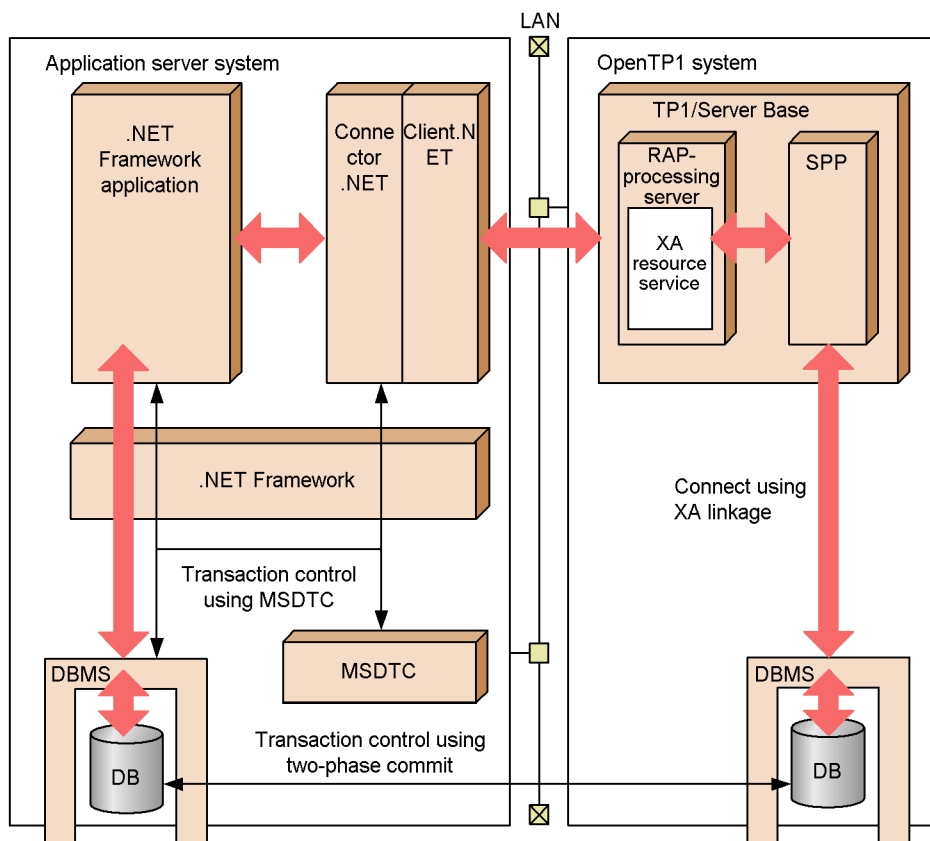
#1: J2EE Connector Architecture  
 #2: uCosminexus TP1 Connector or Cosminexus TP1 Connector

**(b) Linkage with a .NET Framework application**

A .NET Framework application uses MSDTC to control transactions, and sends instructions to OpenTP1 via Connector .NET. OpenTP1 receives transaction requests from the .NET Framework application on the RAP-processing server, and processes transactions using the XA resource service

Figure 3-9 shows the flow of transaction control when OpenTP1 is linked with a .NET Framework application.

Figure 3-9: Transaction control when linked with a .NET Framework application



## (2) XA resource interface and J2EE Connector Architecture

Control of OpenTP1 transactions from a J2EE application server conforms to J2EE Connector Architecture. J2EE Connector Architecture is the standard specifications for connecting an application server and a resource adapter.

By conforming to J2EE Connector Architecture, the application server can control the OpenTP1 transactions in the same manner it handles a DBMS and other resource adapters. OpenTP1 can function as a standard resource adapter on different types of application servers.

There are two transaction control interfaces conforming to J2EE Connector Architecture:

- XA resource interface

- Local transaction interface

For details about each interface, see the J2EE Connector Architecture documentation.

The XA resource service is used to accept transaction control using the XA resource interface on OpenTP1.

### **(3) Prerequisite facilities for using the XA resource service**

To use the XA resource service, you must use the remote API facility and the following products:

When linking with a J2EE application server

- TP1/Client/J
- uCosminexus TP1 Connector or Cosminexus TP1 Connector

When linking with a .NET Framework application

- Client .NET
- Connector .NET

When linking with a .NET Framework application, we recommend that you specify 00000001 in the `value` attribute of the `<extendLevel>` element in the TP1/Client for .NET Framework configuration definition. By specifying 00000001, the IP address of the node requesting the transaction will be shown in message KFCA32045-E, which appears when transaction determination fails because the XAR file has insufficient record length.

#### **(a) Remote API facility**

The *remote API facility* is the process by which an API issued by the client side is forwarded by OpenTP1 to the server, and server processes are used to execute the API instead of the client. The server that executes API calls for the client is called the *RAP-processing server*.

The XA resource service runs on the RAP-processing server that uses the remote API facility. The RAP-processing server manages all transaction requests from an application server linked by the XA resource service. Therefore, to use the XA resource service, the RAP-processing server must be started beforehand. If the RAP-processing server is not running, all transaction requests from the application server linked by the XA resource service will result in an error.

OpenTP1 sets up a logical communication path called a *permanent connection* between the UAP that requests remote API calls and the RAP-processing server. There are two scheduling methods for the permanent connection: *static connection schedule mode* and *dynamic connection schedule mode*. The XA resource service can use either mode.

For details about the remote API facility, see *3.7 Remote API facility*.



**(b) TP1/Client/J or Client .NET**

*TP1/Client/J* is a software product that acts as the intermediary when a transaction request is transferred from a J2EE application server to OpenTP1.

*Client .NET* is a software product that acts as the intermediary when a transaction request is transferred from a .NET Framework application to OpenTP1.

The XA resource service requires the remote API facility. For this reason, service requests by RPCs from TP1/Client/J or Client .NET must be implemented using the remote API facility. Also note that the available connect mode is limited to the auto connect mode when the remote API facility is used.

TP1/Client/J and Client .NET use four types of remote procedure service calls: Synchronous response, non-response, chained, and not inheriting transactions. The XA resource service can use any of the four.

In a normal RPC, multiple host names can be specified as OpenTP1 nodes receiving the transaction request. When using the XA resource service, however, only one OpenTP1 host name can be specified to receive the transaction request from the application server linked by the XA resource service.

Table 3-1 lists the TP1/Client/J or Client .NET functions that can be used with the XA resource service.

*Table 3-1:* List of TP1/Client/J or Client .NET functions supported by the XA resource service

TP1/Client/J or Client .NET function	Existing option	Availability
RPC service request method	Remote API facility	Yes
	Scheduler direct facility	No
	Name service facility	No
Connect mode	Auto connect mode	Yes
	Non-auto connect mode	No
RPC service call type	Synchronous response	Yes
	Non-response	Yes
	Chained	Yes
	Not inheriting transactions	Yes
Specification of OpenTP1 host name as the receiver (specified in the <code>dchost</code> operand)	One name only	Yes
User data compression	Data compression	Yes

Legend:

Yes: Available

No: Not available

For details about TP1/Client/J functions, see the manual *OpenTP1 TP1/Client User's Guide TP1/Client/J*.

For details about Client.NET functions, see the *OpenTP1 TP1/Client for .NET Framework User's Guide*.

**(c) uCosminexus TP1 Connector or Cosminexus TP1 Connector**

*uCosminexus TP1 Connector* or *Cosminexus TP1 Connector* is a software product for controlling OpenTP1 communication and transactions, acting as a resource adapter compliant with J2EE Connector Architecture. An application server running on J2EE can instruct OpenTP1 to perform a two-phase commit by controlling transactions to uCosminexus TP1 Connector or Cosminexus TP1 Connector.

For precautions when linking OpenTP1 with a J2EE application server using the XA resource service, see the uCosminexus TP1 Connector or Cosminexus TP1 Connector documentation.

**(d) Connector .NET**

*Connector .NET* is a software product for controlling communication and transactions from a .NET Framework environment to OpenTP1. In MSDTC linkage, Connector .NET works as a resource manager participating in the MSDTC transaction. MSDTC can instruct OpenTP1 to perform a two-phase commit by controlling transactions to Connector .NET.

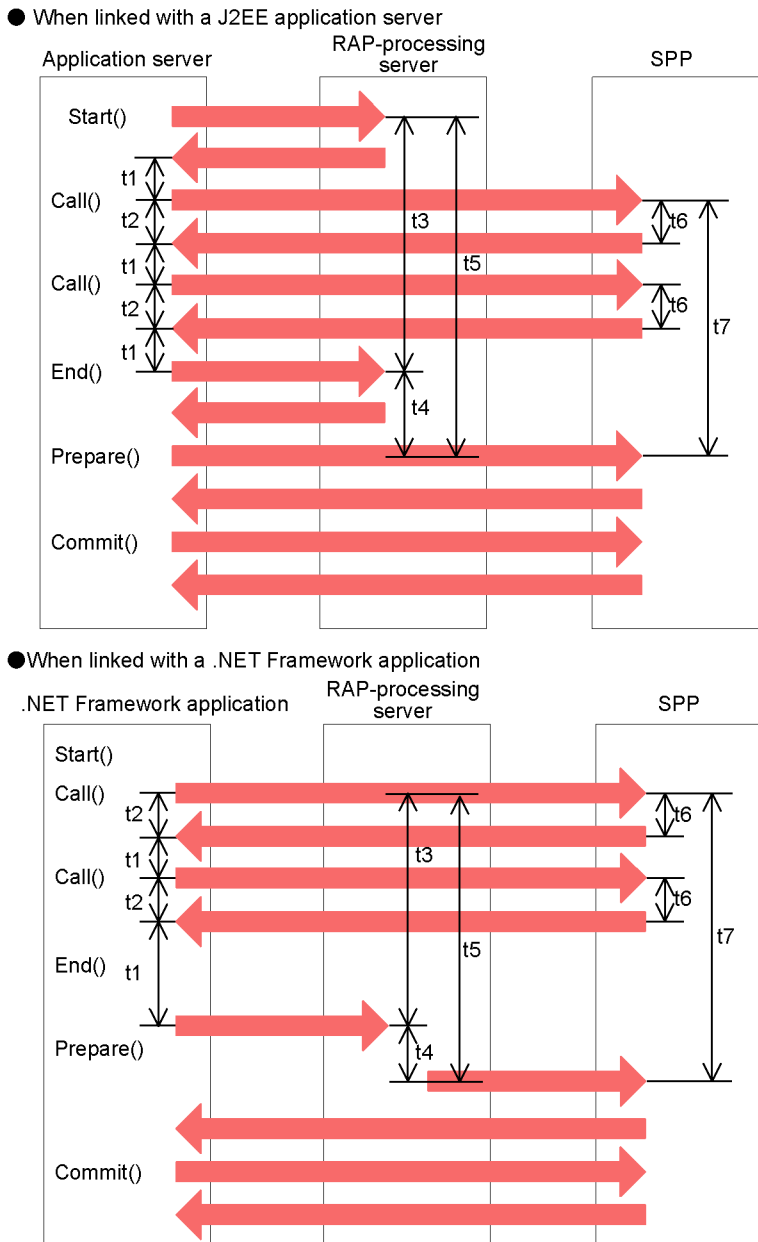
For precautions when linking OpenTP1 with a .NET Framework application using the XA resource service, see the *OpenTP1 TP1/Connector for .NET Framework User's Guide*.

**(4) Timer monitoring facility**

Various timer monitoring facilities are available for canceling transaction processing when a request from an application server linked by the XA resource service stops, or when processing of a user application program is held up.

Figure 3-10 describes the range of timer monitoring facilities that apply to an application server linked by the XA resource service, an RAP-processing server, and an SPP.

Figure 3-10: Ranges of the timer monitoring facilities



Note: Start() to Commit() on the application server side are internal OpenTP1 functions.

Table 3-2 lists the types of monitoring performed, and the required operand, for timer

monitoring periods t1 to t7 in the figure.

Table 3-2: Types of timer monitoring and applicable operand

Period	Monitored item	J2EE application server#	.NET Framework application	OpenTP1
t1	Inquiry interval	TP1/Client/J environment definition: dcltinquiretime operand	TP1/Client for .NET Framework configuration definition: inquireTime attribute of the <rapService> element	RAP-processing listener service definition: rap_inquire_time operand
t2	Message exchange	TP1/Client/J environment definition: dcwatchtim operand	TP1/Client for .NET Framework configuration definition: watchTime attribute of the <rpc> element	RAP-processing listener service definition, user service default definition, or system common definition: watch_time operand
t3	Transaction branch processing	TP1/Client/J environment definition: dclttrexpmtm operand	TP1/Client for .NET Framework configuration definition: expireTime attribute of the <xarTransaction> element	RAP-processing listener service definition, user service default definition, or transaction service definition: trn_expiration_time operand
t4	Transaction branch in idle state	--	--	XA resource service definition: xar_session_time operand
t5	Completion of transaction branch processing	--	--	RAP-processing listener service definition, user service default definition, or transaction service definition: trn_completion_limit_time operand
t6	Transaction branch processing	--	--	User service definition, user service default definition, or transaction service definition: trn_expiration_time operand

Period	Monitored item	J2EE application server#	.NET Framework application	OpenTP1
	Processing of a service function	--	--	User service definition, user service default definition: service_expiration_time operand
t7	Completion of transaction branch processing	--	--	User service definition, user service default definition, or transaction service definition: trn_completion_limit_time operand

Legend:

--: Not applicable.

#

For details about the definition, see the manual *OpenTP1 TPI/Client User's Guide TPI/Client/J*.

### 3.1.8 XA interface

The XA interface is a part of a DTP model defined by X/Open.

The XA interface defines how the transaction manager and resource manager communicate with each other. UAPs do not use the XA interface directly. The following table lists the XA library subroutines used by OpenTP1 and the resource manager.

Table 3-3: XA library subroutines

No.	Function name	Description
1	ax_reg	Registers the resource manager in the transaction manager.
2	ax_unreg	Unregisters the resource manager from the transaction manager.
3	xa_close	Closes the resource manager.
4	xa_commit	This function notifies the resource manager that all the resource managers will be committed after they are found to be committable (in the <i>prepare</i> status).
5	xa_end	Notifies the resource manager that the transaction has ended.
6	xa_forget	Notifies the resource manager that the information about the ended transaction can be discarded at the discretion of the resource manager.
7	xa_open	Opens the resource manager.

### 3. Functions

<b>No.</b>	<b>Function name</b>	<b>Description</b>
8	<code>xa_prepare</code>	Notifies the resource manager that the transaction is being committed.
9	<code>xa_recover</code>	Obtains a list of undetermined transactions in the resource manager.
10	<code>xa_rollback</code>	Notifies the resource manager that the transaction is to be rolled back.
11	<code>xa_start</code>	Notifies the resource manager that the transaction was started or restarted

---

## 3.2 Processing in an OpenTP1 client/server configuration

---

OpenTP1 supports three different kinds of client/server communications that:

- use OpenTP1's remote procedure calls (RPCs)

This is a communication between UAP processes that uses OpenTP1's library functions. A UAP uses functions to issue service requests to a remote UAP. Normal RPC communication uses service information for the server UAPs that are managed by the name service at each node. The name service at each node manages the nodes in the network containing the target server UAP by exchanging service information. The *optional function for service information searches* enables you to choose a service request method that is appropriate to the communication mode in use. If OpenTP1-to-OpenTP1 communication is disabled by a network failure, you can prevent the subsequent RPC from resulting in an error by using *node management*.

For details, see the following:

- Overview of RPCs: *3.2.1 Communication via RPCs that use the OpenTP1 library*
- Optional function for service information searches: *3.2.2 Optional function for service information searches*
- Node management: *3.2.3 Node management in OpenTP1*
- Flow of RPC processing: *D. Overview of Remote Procedure Call Processing*

- use the X/Open XATMI interface

This communication method complies with the DTP model stipulated by X/Open.

- use the X/Open TxRPC interface

This communication method supports DCE RPC with a transaction facility that is stipulated by X/Open. TxRPC communications directly call user-created functions.

If *TCP/IP* is used as a communication protocol, all of the above client/server configuration communications can be performed. If *OSI TP* is used as a communication protocol, communications use the *XATMI interface*. To perform OSI TP communications, you need TP1/NET/OSI-TP-Extended in your OpenTP1 system. For details about the OSI TP communications, see *.3.6 Client/server communications using OSI TP*.

### 3.2.1 Communication via RPCs that use the OpenTP1 library

This subsection describes client/server communications via RPCs that use OpenTP1

library functions.

### (1) How RPCs are used to request a service

A service is a unit of processing called by an RPC and is the facility that a server UAP provides to a client UAP. A server UAP can execute more than one service. The group of services provided by one server UAP is called a *service group*.

When a programmer codes a client UAP so that it can make a request for a service from a service-providing program, the programmer must specify in the RPC function parameters (e.g., `dc_rpc_call()` parameters) the various RPC items required for the service request. Among these items are the *service-group name* and the *service name*. The programmer, however, does not need to code server network addresses into the parameters because the OpenTP1 *name service* manages the service names and network addresses of nodes containing server UAPs.

### (2) Response RPCs and no-response RPCs

RPCs are categorized as either a *response-type RPC* or a *non-response RPC*, according to whether a response is received. Figure 3-11 illustrates RPC types.

Figure 3-11: RPC types

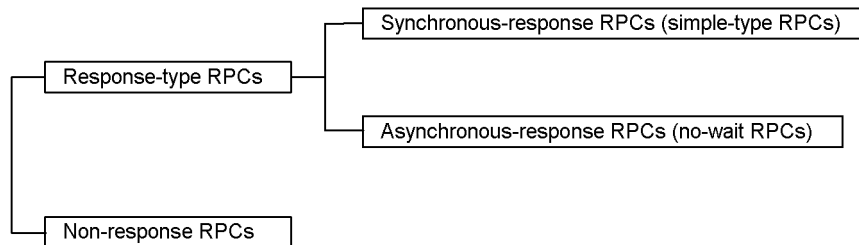
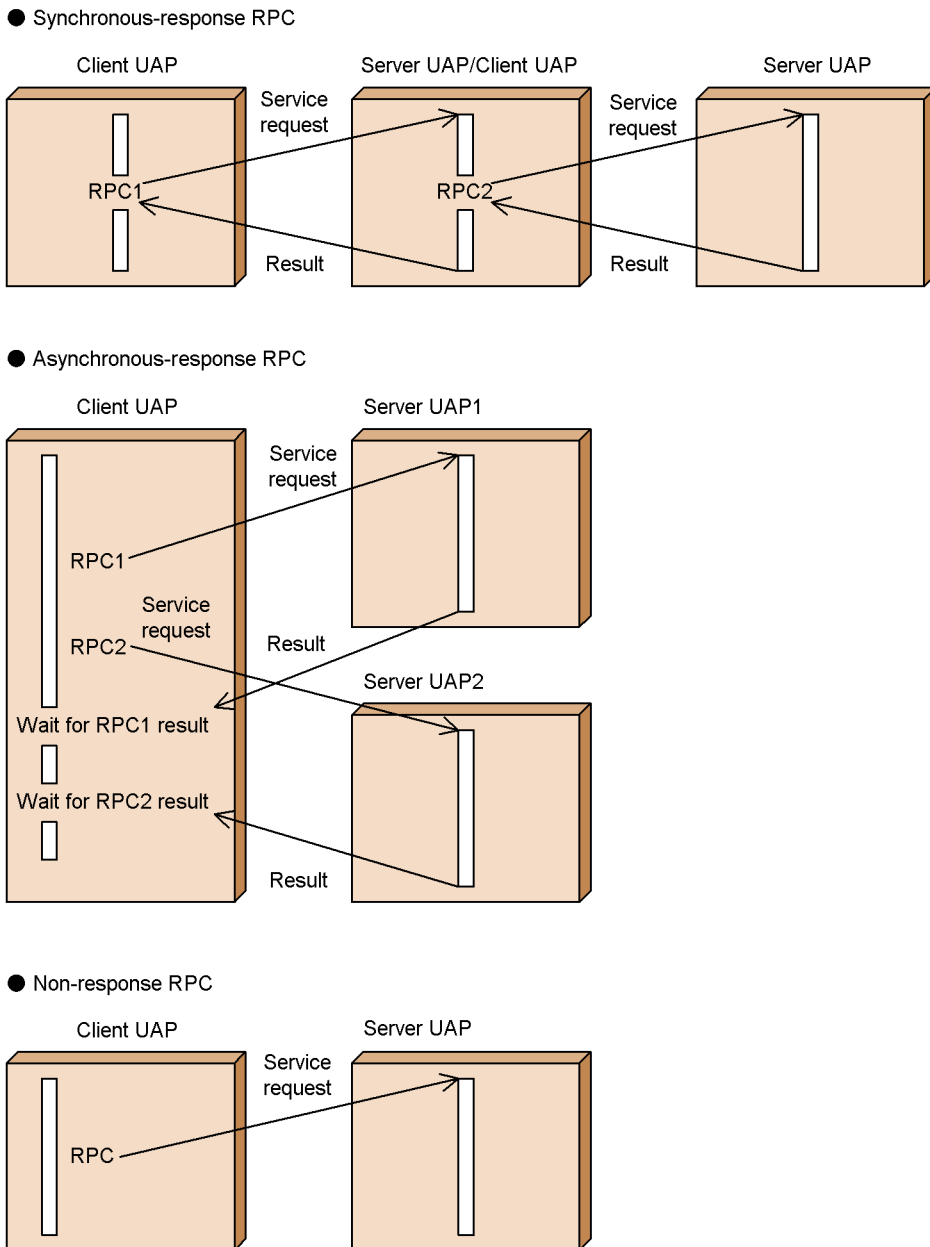


Figure 3-12 summarizes the processing for each RPC type.



Figure 3-12: Overview of processing for each RPC type



**(a) Synchronous-response RPCs**

Response RPCs, which consist of synchronous-response RPCs (simple RPCs) and

asynchronous-response RPCs (no-wait RPCs)

**(b) Asynchronous-response RPCs**

After making a service request, an asynchronous-response RPC continues processing without waiting for a response. A function (`dc_rpc_poll_any_replies`) must be issued to receive a response. This function waits for a response, and returns an error if the response wait time set in the function is exceeded.

Asynchronous-response RPCs are also known as *no-wait RPCs*.

**(c) Non-response RPCs**

A non-response RPC is an RPC to which no processing result is returned. No response is received when a service request is made using a non-response RPC. The UAP that made the request continues processing.

Further services can be requested of a server UAP after it has received one service request (thus creating an *RPC nest*).

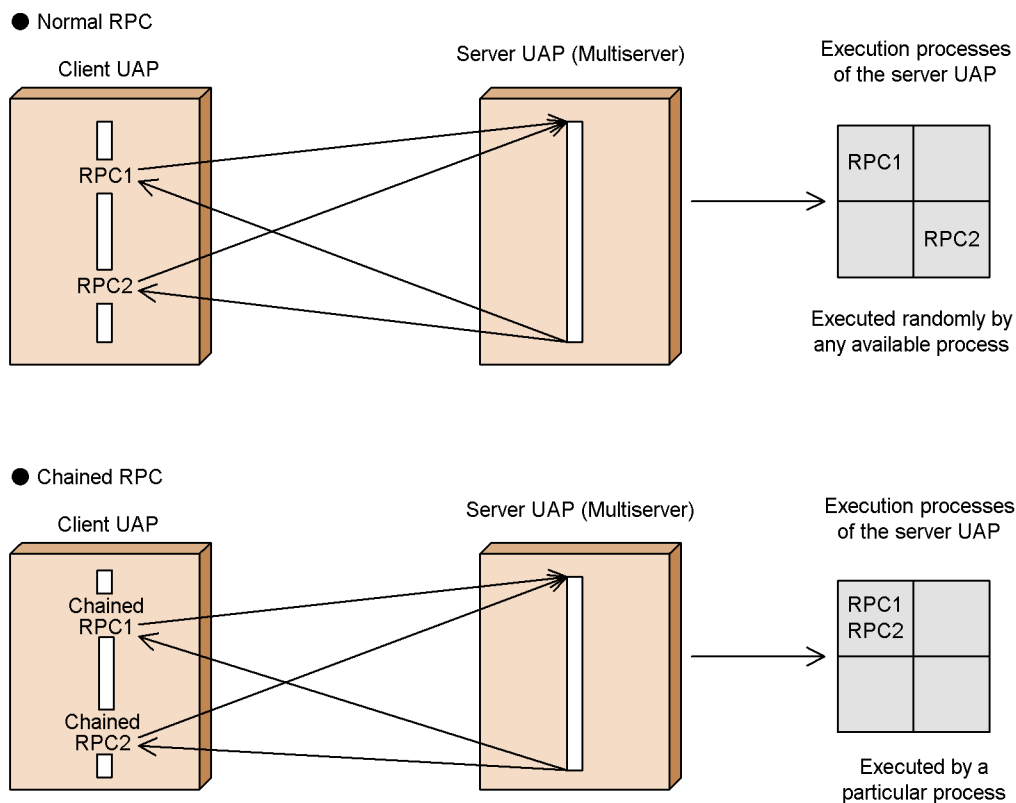
**(3) Chaining RPCs**

The OpenTP1 *Multiserver facility* can reduce the time it takes to process multiple requests for a service. The Multiserver facility enables the same server UAP to start multiple processes at the same time.

A possible drawback to this facility is that a process is started for each service of a server UAP that uses the Multiserver facility; thus, when a client UAP calls the same service group two or more times, the server UAP of that service group might not be executed in the same process as previously. OpenTP1, however, can handle this problem by preventing an unnecessary increase in the number of processes, and thereby reduce the load required for processing a transaction. OpenTP1 does this by using *chained RPCs*, which are a group of RPCs using the same process. Chaining is specified in the `dc_rpc_call()` parameters. A programmer can use chained RPCs so that when a synchronous-response RPC requests the services belonging to the same service group two or more times, OpenTP1 executes the service by the same process.

Figure 3-13 compares a normal RPC with a chained RPC.

Figure 3-13: Comparison of a normal RPC and a chained RPC



The use of chained RPCs can reduce the number of user processes required for processing one transaction and can reduce the load required for transaction processing. When executing chained RPCs as a transaction, operate the chained RPCs in one transaction branch. (Global transactions are described in *3.1 Transaction Control*.)

Chained RPCs are guaranteed for each UAP process. Even in the same global transaction, if the client UAPs are different, it can not be guaranteed that a service that is called multiple times will be started in the same process.

#### (a) Time monitoring of chained RPCs

In a UAP from which a service is requested, OpenTP1 can monitor chained RPCs from the time a response is returned to the client UAP, until the time the next service request comes or until the time that transaction synchronization point processing occurs. In the user service definition, you can specify a limit for this monitoring time. If the monitored time exceeds the limit or if the next request for a service or for synchronization point processing does not come, OpenTP1 assumes that an error

occurred in the client UAP and the server UAP abnormally terminates.

**(4) Compressing the send data for remote procedure calls**

To reduce the LAN network load, RPC can compress the send and receive data. For compressing data, specify the following values in the system definitions.

■ TP1/Server Base

Specify a Y for the `rpc_datacomp` operand in the system common definition.

■ TP1/Client

Specify data compression for the client environment definition.

If the system at the client specifies the data compression, OpenTP1 of the server automatically restores the compressed data to process regardless of the `rpc_datacomp` operand specification. After that, OpenTP1 re-compresses the data and returns a response to the client.

Even if the network load is reduced by the send data compression, the communication time may be lengthy because of compression and decompression of data in a node. Determine whether data compression is specified or not, according to the job types and the communication configurations.

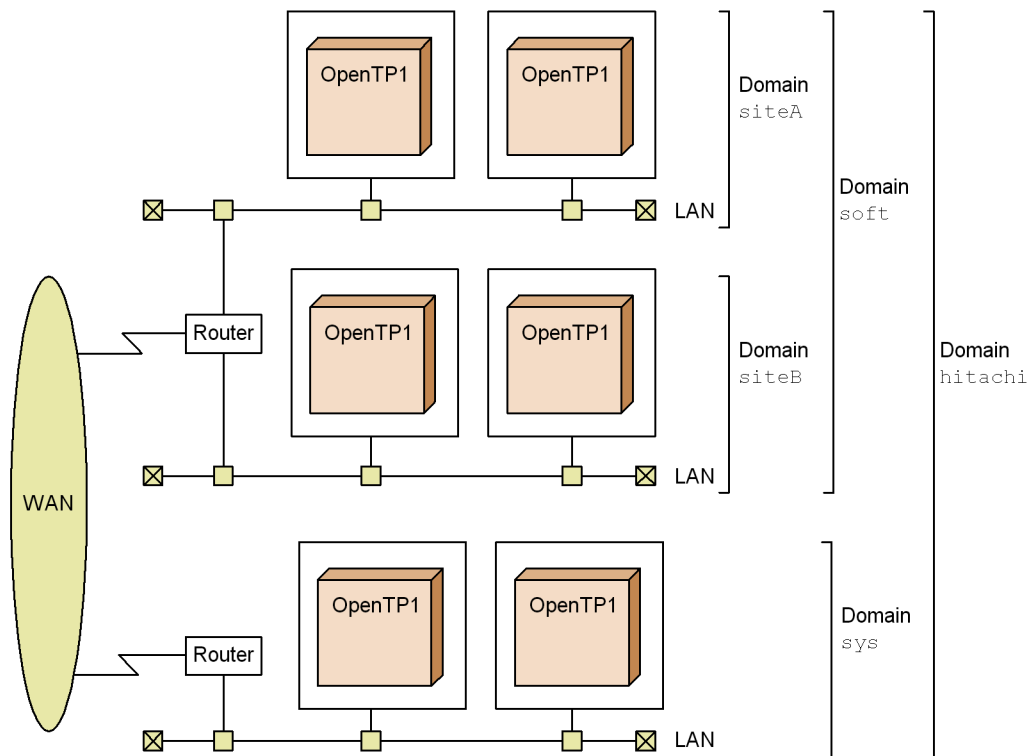
**(5) Service request method in a large-scale distributed system**

**(a) Overview of the domain name system**

In a network system via a WAN or an inter-department system of a company, searching for a name or managing the system names is time-consuming work. When using remote procedure calls in such a large-scale distributed system, the user can solve these problems by configuring a domain name system (DNS).

In a domain name system, the user does not have to manage the entire system and can manage it in groups as domains. Systems of departments are domains. Service names and other names are managed within each domain individually. Therefore, the user does not have to manage the names in the entire system. Figure 3-14 shows an overview of domain-based management.

Figure 3-14: Overview of domain-based management



## Description:

To use a service in another domain, specify domain names in an argument of the `dc_rpc_call` function. If a service is requested from the client in the domain `sys` with specification of domain names `siteA`, `soft`, and `hitachi`, the domains are queried in order of `hitachi`, `soft`, and `siteA`.

**(b) Service request method**

The following describes how to request a service.

## Specifying a service group name and service name

Normally, to request a service, specify a service group name and service name in the `dc_rpc_call()` function.

For a large-scale distributed system, you cannot use a service across domains by specifying a service group name and service name in the `dc_rpc_call()` function.

## Specifying a service group name + domain name and service name

To use a service across domains in a large-scale distributed system, specify *service group name + domain name* and *service name* in the `dc_rpc_call()`

function.

When the domain name is added to the service group name in this function, the service request is passed to a *domain-alternate schedule service* that has access to the requested domain. The domain-alternate schedule service allocates a request to a server UAP among the servers of the domain.

The request is allocated to a server UAP of the specified domain and is not allocated to a server UAP of another domain. When the domain-alternate schedule service is used, the name service does not need to manage the network addresses of all the nodes.

When the server UAP is a server that receives requests from the socket, you cannot add the domain name to the service group name in the `dc_rpc_call()` function. For details of the server that receives requests from the socket, see *3.4.1(1) Schedule queues for SPPs*.

#### Specifying the destination of a service request

While the `dc_rpc_call()` function does not recognize a location, a user can identify the location of a desired server for requesting a service, and then use the service with the `dc_rpc_call_to()` function. The following describes how to specify the destination to issue a service request.

- Specifying the host name

This method specifies the machine where the desired service exists by specifying the host name that is specified in `/etc/hosts` for network management. To use this method, the specification of `name_port` of the system common definition must be the same for the OpenTP1 system in the specified host and for the OpenTP1 system that issued the `dc_rpc_call_to()` function.

- Specifying the node identifier

This method specifies the OpenTP1 system by specifying the node identifier that is specified in `node_id` of the system common definition.

To use this method, the global domain<sup>#</sup> must contain the host name of the OpenTP1 node to which the service is requested, with the node identifier specified.

#

The global domain means a set of the following node names:

- When `N` is specified in the `name_domain_file_use` operand of the system common definition

The global domain is the set 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 of the system common definition

The global domain is the set of node names specified in the domain definition files. The domain definition files are stored at the following locations:

Domain definition file for `all_node`:

- Under the `$DCCONFPATH/dcnamnd` directory

Domain definition file for `all_node_ex`:

- Under the `$DCCONFPATH/dcnamndex` directory

- Specifying the host name + port number

This method specifies the service request destination by specifying:

- The host name that is specified in `/etc/hosts`.

- Port number of the name service that is specified in `name_port` of the system common definition for the OpenTP1 system contained in the host specified above.

The specification of the `name_port` operand of the system to which the service is requested may differ from the specification of the `name_port` operand for the system that issued the service request.

This facility requires TP1/Extension 1 installed. If TP1/Extension 1 is not installed, the operation is not guaranteed.

### **(c) Setting up the environment of a large-scale distributed system**

The nodes that make up a domain must be specified by `all_node` in the system common definition.

To specify the domain-alternate schedule service, specify, in the `namdomainsetup` command, the name of the host where the domain-alternate schedule service exists. Then, specify, in `/etc/services`, the port number of the domain-alternate schedule service and a service name `OpenTP1scd`.

When using a domain name system, more time may be needed to complete the system startup because each node address is inquired during startup of the server system. In this case, specify a domain name and host name beforehand in the `domain_masters_addr` operand and specify a port number of the domain-alternate schedule service to the `domain_masters_port` operand in the system common definition. Specifying these operands reduces the time it takes to inquire about each node address while starting a service.

### **(6) How to request a service without using a name service**

The normal RPC uses the service information of the server UAP managed by the name

service of each node to perform communication. The name services of nodes exchange service information to manage at which node of the network the target server UAP exists. In a large-scale system connecting many nodes, the exchange of service information by name services may tax the network.

When you have defined service information in a definition file, you can execute an RPC without using a name service. This method does not require querying the name service for service information to reduce the load on the network.

Define the service group name and service information (host name and port number) in the *user service network definition file*. Specify whether to use the name service or the service information in the definition file with the `rpc_destination_mode` operand of the user service definition at issuance of the function `dc_rpc_call()`. When the function `dc_rpc_call()` is called from a UAP for which the user service definition specifies using the service information in the definition file, OpenTP1 references the user service network definition file. When the server UAP information is defined, OpenTP1 calls the server UAP by using the definition. When the server UAP information is not defined, OpenTP1 calls the server UAP by using the name service. Note that the server UAP in the user service network definition must be a UAP the user service definition specifies `receive_from=queue` (server that receives requests from the queue).

If you specify multiple service information items (host names) in the user service network definition file, the service request destination is selected at random from the specified hosts, and a service request is sent to the selected host. Once sending of a service request is successful, the subsequent `dc_rpc_call` invocations made in the UAP to the same service group continue to send service requests to the same host.

Note that a destination host is randomly selected again if the destination reselection interval is specified in the `-t` option of the `dcsvgdef` definition command and the following time exceeds the specified interval:

- Time since selection of the current service-request destination host used by multiple `dc_rpc_call` invocations made in the UAP to the same service group

### 3.2.2 Optional function for service information searches

The optional function for service information searches consists of the global search facility and the service information prioritizing function. You can use these functions in the OpenTP1 system to issue service requests that use horizontal distribution and to preferentially use service information for a specific node.

#### (1) *Global search facility*

Using the *global search facility*, you can obtain the following service information:

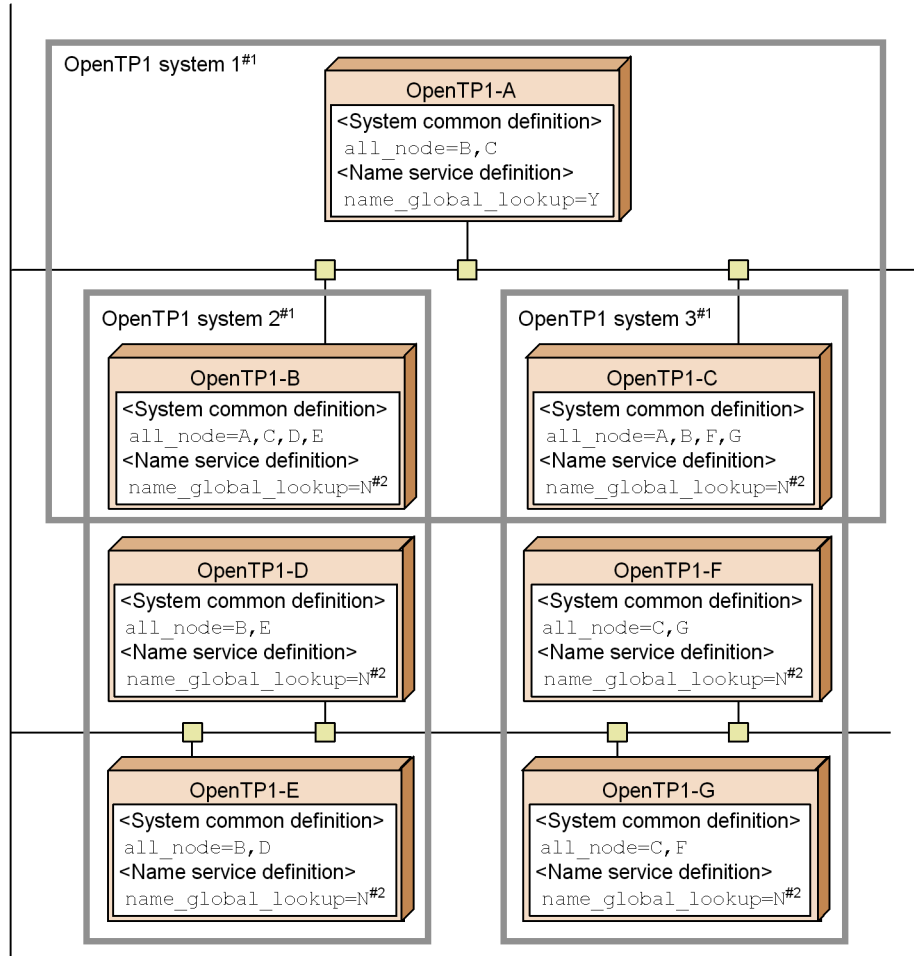
1. Information about services that run on the nodes specified in the `all_node` operand



2. Information about services that run on the nodes specified in each of the `all_node` operands managed by the name service of the nodes referred to in 1. above.

The figure below shows an example of a system configuration using the global search facility. Specify the `name_global_lookup` operand in the name service definition for each OpenTP1 node as shown in the figure. For details about specifying this operand, see the manual *OpenTP1 System Definition*.

Figure 3-15: System configuration when using the global search facility



Note: The version of OpenTP1-B through OpenTP1-G must be 03-02 or later.

- #1: An OpenTP1 system means a group of OpenTP1 nodes that are specified in the all\_node operand at each OpenTP1 node.
- #2: An OpenTP1 node for which N is specified in the name\_global\_lookup operand or an OpenTP1 node that does not support the name\_global\_lookup operand is valid.

The search range from OpenTP1-A encompasses the OpenTP1 nodes specified in the all\_node operands for OpenTP1-B and C. That is, OpenTP1-A can communicate with services in OpenTP1 systems 2 and 3 as well as those in OpenTP1 system 1. The nodes specified in the all\_node operands for OpenTP1-D, E, F and G are not included in the search range from OpenTP1-A.

When the global search facility is enabled, the `dc_rpc_call_to` function cannot be used with the port number of the name service specified in the `portno` argument of the `DCRPC_BINDTBL_SET` function.

To use the `dc_rpc_call_to` function with a node ID specified in the `nid` argument of the `DCRPC_BINDTBL_SET` function, you must define the same node ID for all OpenTP1 nodes in the search range of the global search facility (OpenTP1 systems 1 to 3 in Figure 3-15).

Because service information (shutdown status, load status, and so on) is not reported to the OpenTP1 instance that requested the search (OpenTP1-A in Figure 3-15), we recommend that service requests be handled in a parallel distribution, by ensuring that the `all_node` operand for each node in the same system defines the other nodes in that system. Thus, in the figure, OpenTP1-D and E are reciprocally defined in OpenTP1 system 2, as are OpenTP1-F and G in OpenTP1 system 3.

When calculating the value to set in the `name_cache_size` operand in the name service definition, as well as the number of service information items contained in searches requested from the local node, you must also count the number of service information items cached on the nodes specified in the `all_node` operand.

When a service request is sent from TP1/Client/P, TP1/Client/W or TP1/Client/J to an OpenTP1 instance that uses the global search facility (OpenTP1-A in Figure 3-15), information can be collected not only from OpenTP1 system 1 but also from OpenTP1 systems 2 and 3.

## **(2) Service information prioritizing function**

The *service information prioritizing function* prioritizes the service information for a specific node when the name service returns service information to a client UAP that requested services. The node whose service information is returned preferentially is called the *priority selection node*.

Normally, when an RPC is executed (except under special circumstances, such as when there is a high workload) while there are multiple server UAPs that can process a client UAP's service request, those server UAPs are all treated with the same priority. Therefore, a service information search request is sent to the name services at all nodes specified in the `all_node` operand in the system common definition and then all the service information acquired from the responses is returned to the client UAP. The client UAP then selects from the returned service information the server UAP that is to be the target of the RPC, and then executes the RPC.

On the other hand, when the service information prioritizing function is used when there are multiple server UAPs that can process a client UAP's service request, this function returns to the client UAP the service information for the priority selection node. The client UAP then executes the RPC on the server UAP at the priority selection node. In the event of a failure (such as UAP shutdown) at the priority selection node, the function returns to the client UAP the service information for a

node other than the priority selection node. The service information prioritizing function enables you to use the server UAP at the priority selection node normally and to use another node in the event of a failure. In this way, you can treat one server UAP as the running system and another as the standby system.

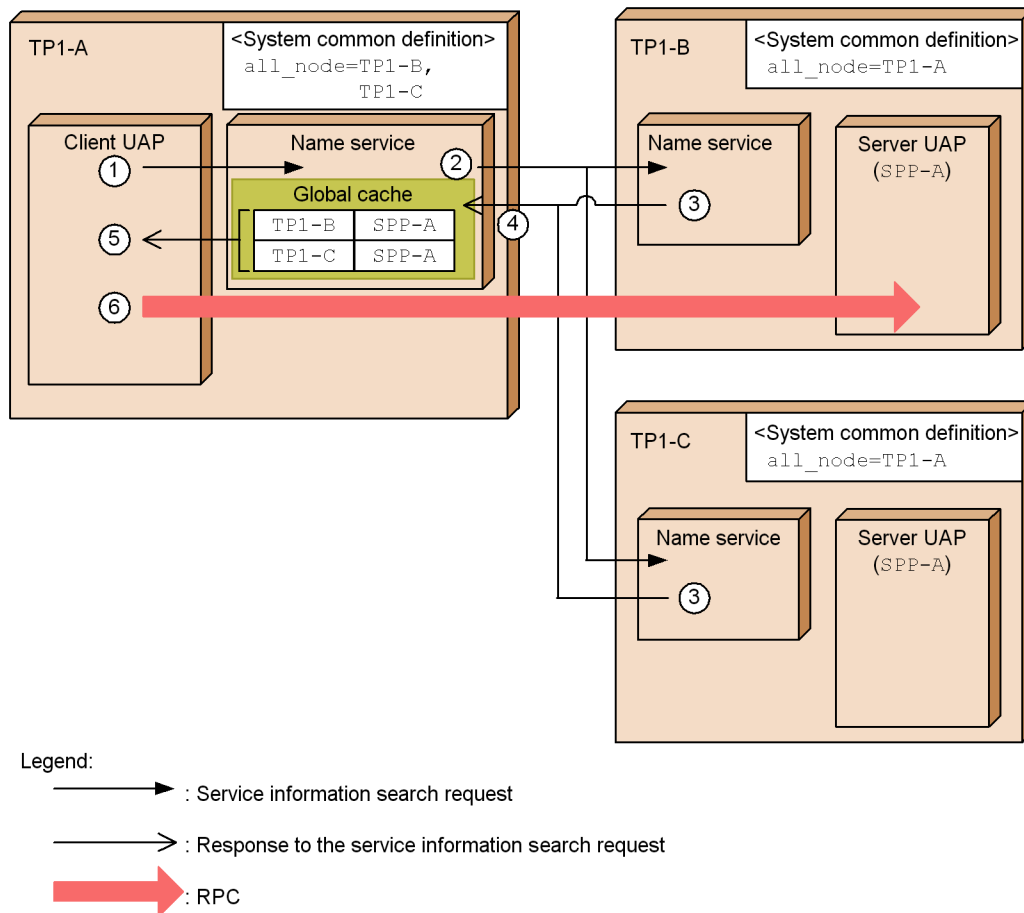
You use the `all_node` operand in the system common definition to specify the priority selection node. The method for specifying the priority selection node definition file is the same as for the domain definition file. For details about how to specify the priority selection node, see the manual *OpenTPI System Definition*.

The following subsections describe a normal RPC flow and an RPC flow when the service information prioritizing function is used. In the descriptions below, the term *global cache* refers to an area used by the name service to manage service information for the servers that are running at other nodes.

- Normal RPC flow

The following figure shows the normal RPC flow.

Figure 3-16: Normal RPC flow



The following describes the RPC flow shown in the figure. The numbers below correspond to the circled numbers in the figure.

1. A client UAP that is ready to execute an RPC issues a service information search request to the name service at TP1-A.
2. The name service sends the service information search request to the name services at TP1-B and TP1-C, which are specified in the `all_node` operand in the system common definition.
3. The name services at TP1-B and TP1-C that receive the search request send the service information for SPP-A running at the local node to the TP1-A that issued the request.

### 3. Functions

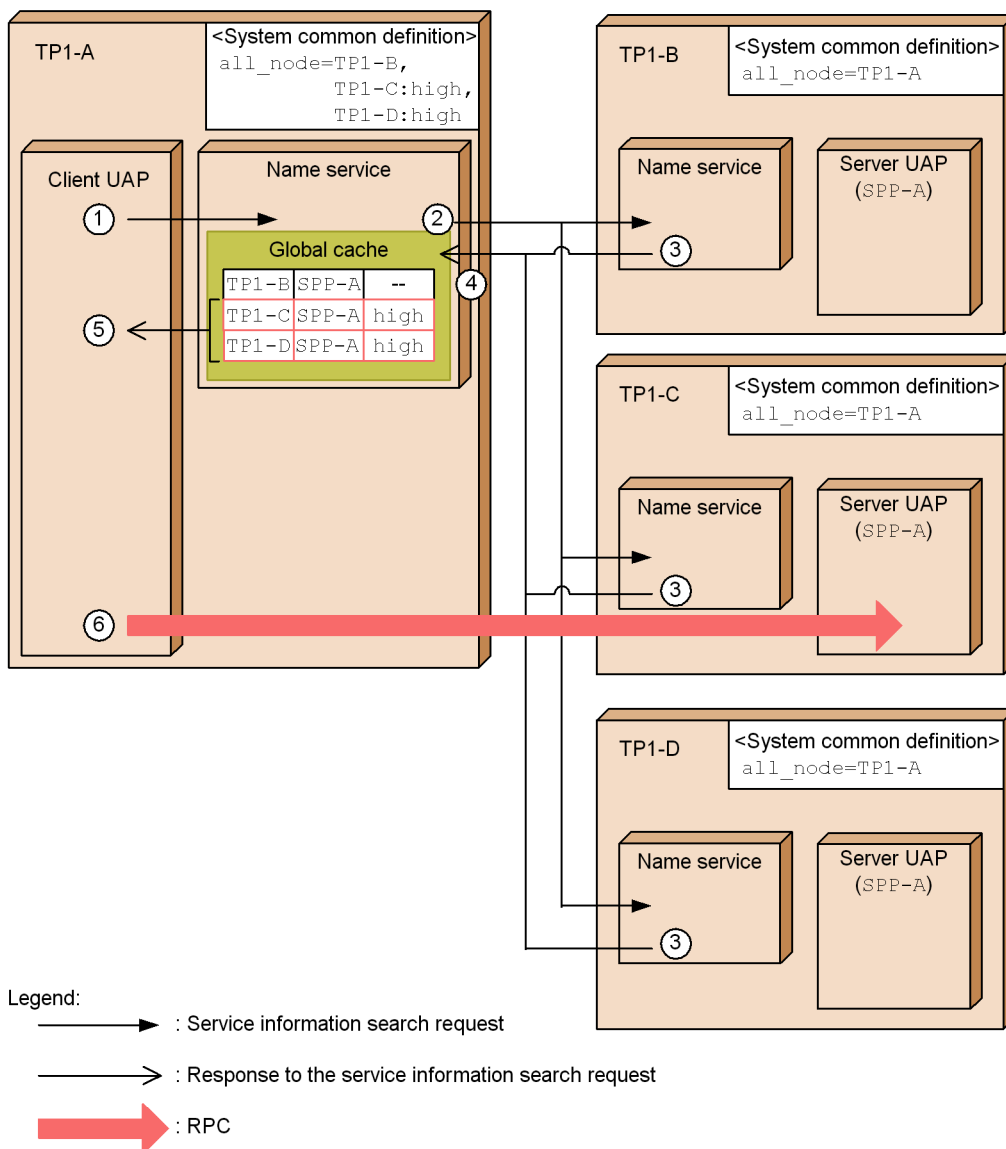
4. The name service at TP1-A registers the received service information in the global cache.
5. The name service returns to the client UAP all service information registered in the global cache.

The client UAP randomly selects the target of the RPC. This example assumes that TP1-B is selected as the target of the RPC.

6. The client UAP executes the RPC on TP1-B.
- RPC flow when the service information prioritizing function is used

The following figure shows the RPC flow when the service information prioritizing function is used. In this example, TP1-C and TP1-D have been specified as priority selection nodes.

Figure 3-17: RPC flow when the service information prioritizing function is used



The following describes the RPC flow shown in the figure. The numbers below correspond to the circled numbers in the figure.

1. A client UAP that is ready to execute an RPC issues a service information search request to the name service at TP1-A.

### 3. Functions

2. The name service sends the service information search request to the name services at TP1-B, TP1-C, and TP1-D, which are specified in the `all_node` operand in the system common definition.
3. The name services at TP1-B, TP1-C, and TP1-D that received the search request send the service information for SPP-A running at the local node to the TP1-A that issued the request.
4. The name service at TP1-A registers the received service information in the global cache.
5. The name service returns to the client UAP the service information for TP1-C and TP1-D only because they have been specified as the priority selection nodes.

This example assumes that from the returned service information, the client UAP selects TP1-C as the target of the RPC.

6. The client UAP executes the RPC on TP1-C.

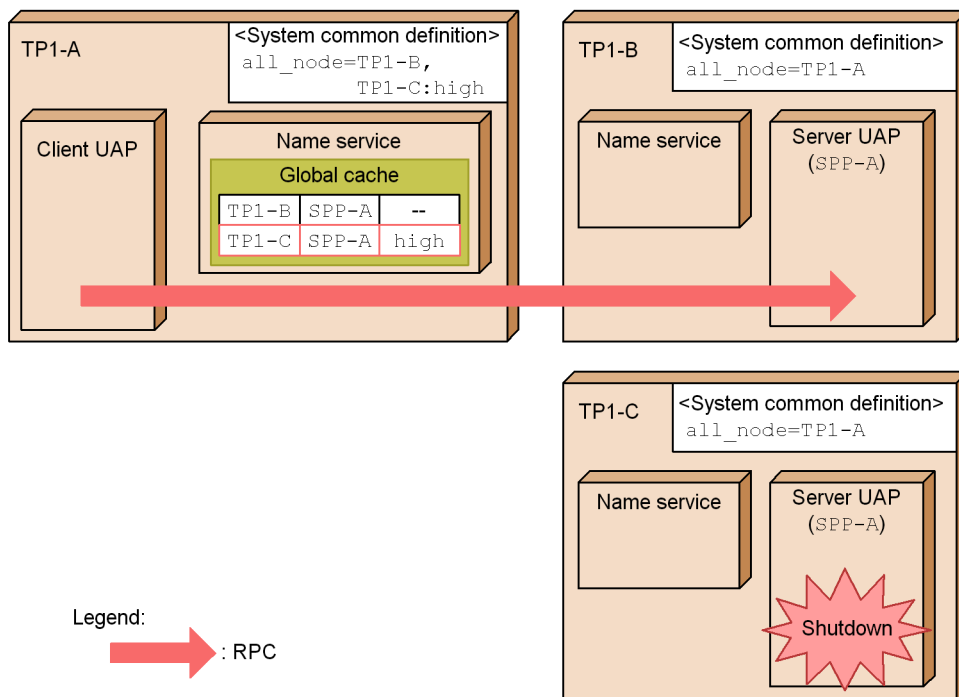
If the server UAP has shut down at the priority selection node or the global search facility is used, the RPC flow differs from the flow shown in the figure above where the service information prioritizing function is used. Such cases are described in (a) through (c) below. Note that name service processing is omitted in the descriptions in the following subsections.

#### **(a) RPC flow when the server UAP has shut down**

The following figure shows an RPC flow when the server UAP at the priority selection node has shut down.



Figure 3-18: RPC flow when the server UAP has shut down



TP1-C has been specified as the priority selection node for TP1-A, but the server UAP at TP1-C has shut down. When the server UAP's shutdown status at TP1-C is sent to TP1-A, TP1-A removes TP1-C as an RPC target candidate. Therefore, TP1-A executes the RPC on the server UAP at TP1-B, even though it is not a priority selection node.

*Note:*

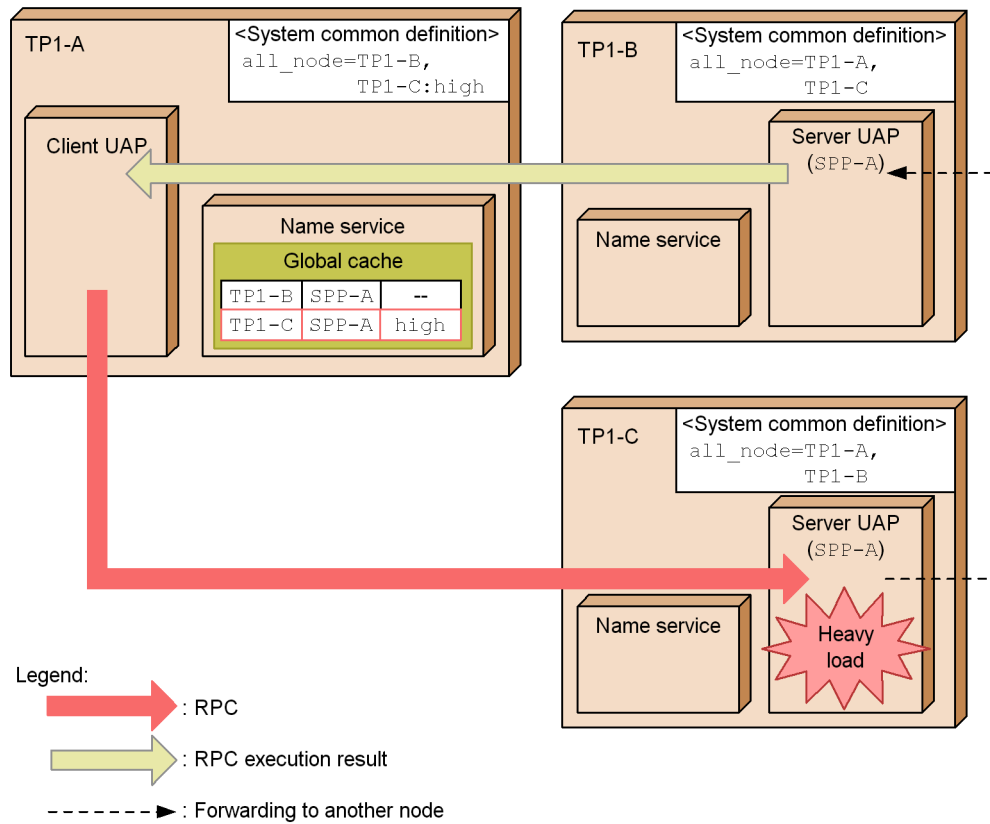
While TP1-A is waiting to be notified of the server UAP's shutdown status at TP1-C, TP1-A executes the RPC on the server UAP at TP1-C. If TP1-B is specified in the `all_node` operand in the system common definition at TP1-C, the RPC might be forwarded from TP1-C to TP1-B. If TP1-B is not specified in the `all_node` operand in the system common definition at TP1-C, the RPC returns an error.

The shutdown information is reported for each server. Therefore, if the server UAP at TP1-C is shut down on a service-by-service basis, the shutdown information is not sent to TP1-A. As a result, TP1-A executes the RPC on the server UAP at TP1-C. If the target service is down, the RPC returns an error.

**(b) RPC flow when there is a heavy workload at the server UAP**

The following figure shows an RPC flow when the server UAP at the priority selection node is sustaining a heavy workload.

Figure 3-19: RPC flow when the server UAP is sustaining a heavy workload



Because TP1-C is specified as the priority selection node at TP1-A, the server UAP running at TP1-C becomes the RPC target candidate. However, the RPC is forwarded to the server UAP at a node specified in the `all_node` operand of the system common definition at TP1-C because the server UAP at TP1-C is sustaining a heavy workload. This example forwards the RPC from TP1-C to the server UAP at TP1-B. The server UAP at TP1-B then executes the RPC for TP1-A.

**(c) RPC flow when the global search facility and the service information prioritizing function are both used**

You can use the global search facility together with the service information prioritizing function. When both functions are used, the RPC depends on whether a global search

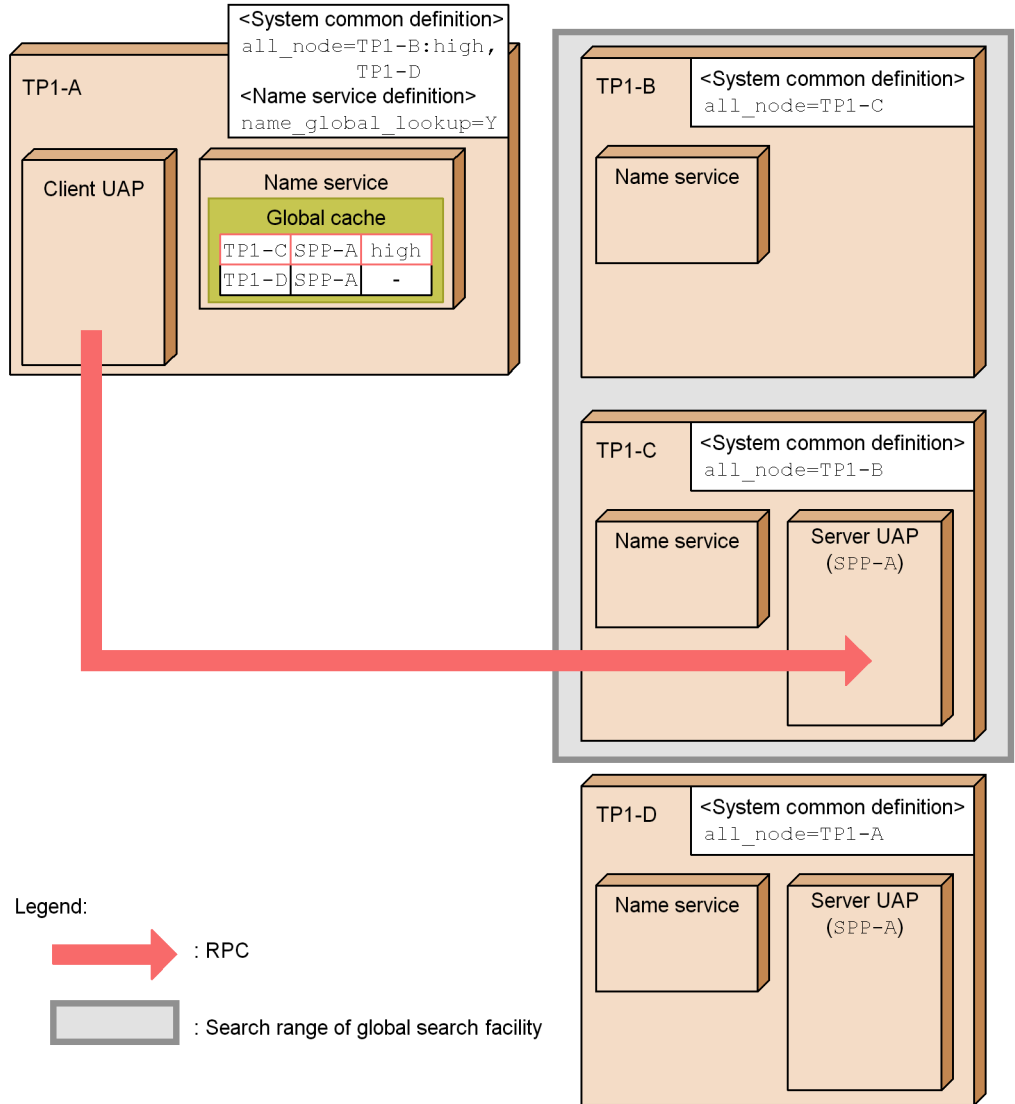
relay node has been specified as the priority selection node.

If a global search relay node has been specified as the priority selection node, the priority selection node specified in the search range of the global search facility becomes the target of the RPC. If no global search relay node is specified as the priority selection node, the priority selection node specified at the requesting node becomes the target of the RPC. Both these cases are described below.

- RPC flow when a global search relay node is specified as the priority selection node

The following figure shows the RPC flow when a global search relay node is specified as the priority selection node.

Figure 3-20: RPC flow when a global search relay node is specified as the priority selection node



Because this example uses the global search facility, TP1-A acquires via TP1-B the service information of TP1-C, which is specified in the `all_node` operand of the system common definition at TP1-B.

Because TP1-B is specified at TP1-A as the priority selection node, the service

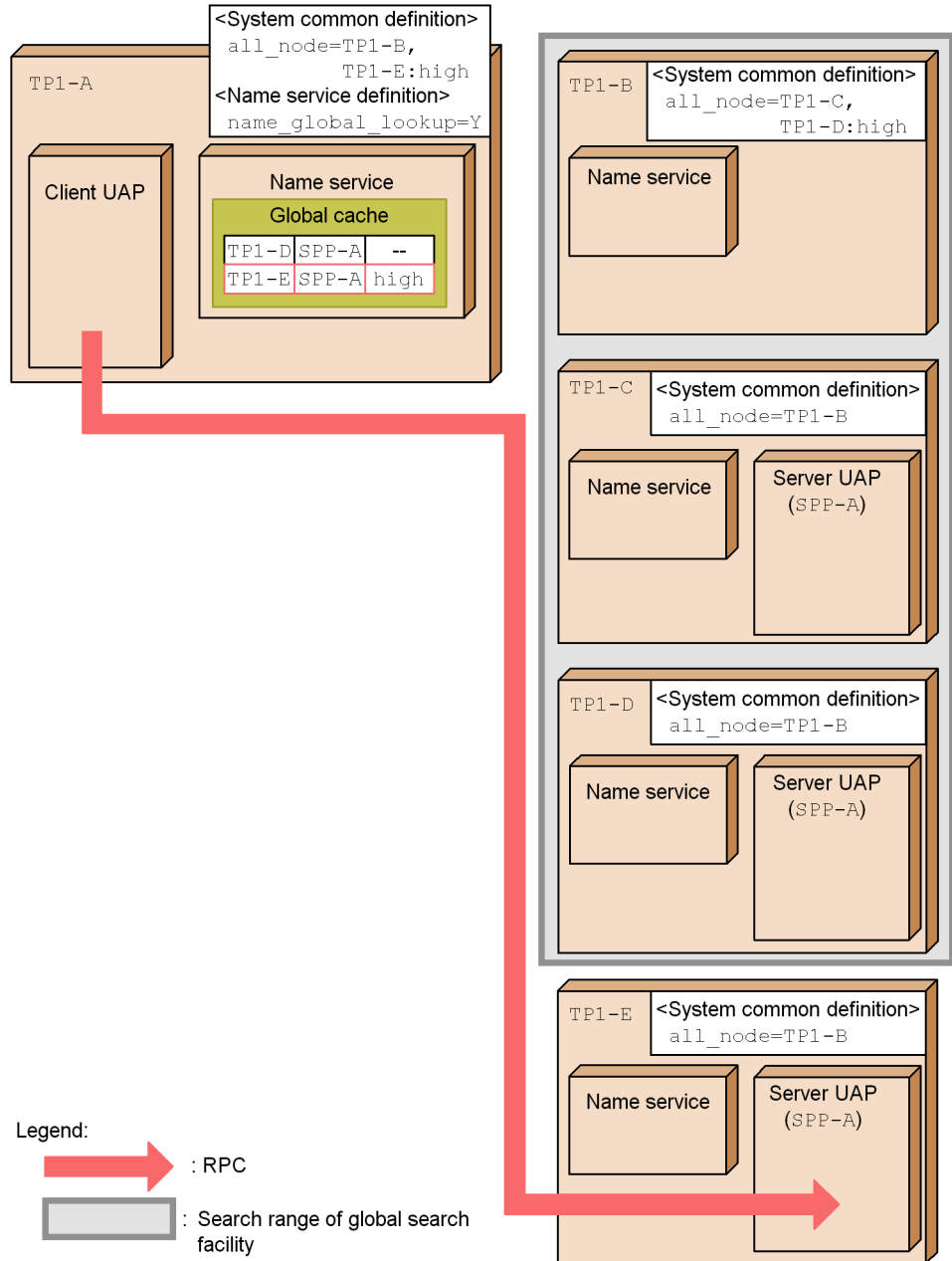
information of TP1-C specified in the `all_node` operand at TP1-B is treated as running at the priority selection node. As a result, TP1-A preferentially selects the service information of TP1-C.

In other words, the service information (service information of TP1-C) that has been acquired via the global search relay node (TP1-B) specified as the priority selection node is selected because it is treated as running at the priority selection node, regardless of the priority at the service requesting node (TP1-A).

- RPC flow when no global search relay node is specified as the priority selection node

The following figure shows the RPC flow when no global search relay node is specified as the priority selection node.

Figure 3-21: RPC flow when no global search relay node is specified as the priority selection node



Because this example uses the global search facility, TP1-A attempts to acquire

via TP1-B the service information of TP1-C and TP1-D, which have been specified in the `all_node` operand of the system common definition at TP1-B. TP1-B selects the service information running at TP1-D, which is a priority selection node at TP1-B, and then returns it to TP1-A.

As a result, TP1-A selects the priority selection node TP1-E.

In other words, the priority at the service requesting node (TP1-A) takes effect. The service information (of TP1-D) acquired via a global search relay node (TP1-B), which is not a priority selection node, is not selected, regardless of the priority at the global search relay node (TP1-B).

#### (d) Notes about using the service information prioritizing function

The following should be noted about using the service information prioritizing function.

- If one or more priority selection nodes are specified and the target server UAP is running at the node where an RPC is to be executed, the node where the RPC is to be executed also becomes a priority selection node.
- The service information prioritizing function is disabled if the `dc_rpc_call_to` function is executed. In such a case, the RPC is executed on the server UAP that is running at the node specified by the host name.
- If all the server UAPs running at the priority selection nodes are shut down, those nodes are removed as priority selection nodes. In such a case, the RPC is executed on the server UAP running at a node that is not specified as a priority selection node.
- If the server UAP running at the priority selection node is sustaining a heavy workload, the RPC is forwarded to the same service group running at a non-priority selection node. Therefore, the RPC might be executed on a server UAP that is running at a non-priority selection node.

For details about handling heavy workloads, see *3.4.3(5) Balancing loads among nodes*.

### 3.2.3 Node management in OpenTP1

Multiple instances of OpenTP1 communicate with one another using RPCs implemented through TCP/IP. TCP/IP establishes a connection between the servers, enabling them to communicate.

If communication fails due to a network error, OpenTP1 instances cannot detect the loss of connection. For this reason, any RPCs issued after a network error may fail. The following describes OpenTP1 facilities for preventing RPC errors after a connection failure.

**(1) Startup notification facility**

When OpenTP1 starts, startup on the local node is reported to the name service of the OpenTP1 instances running on another node, and the connection already established is forcibly closed. This functionality can be used at system switchover, for example.

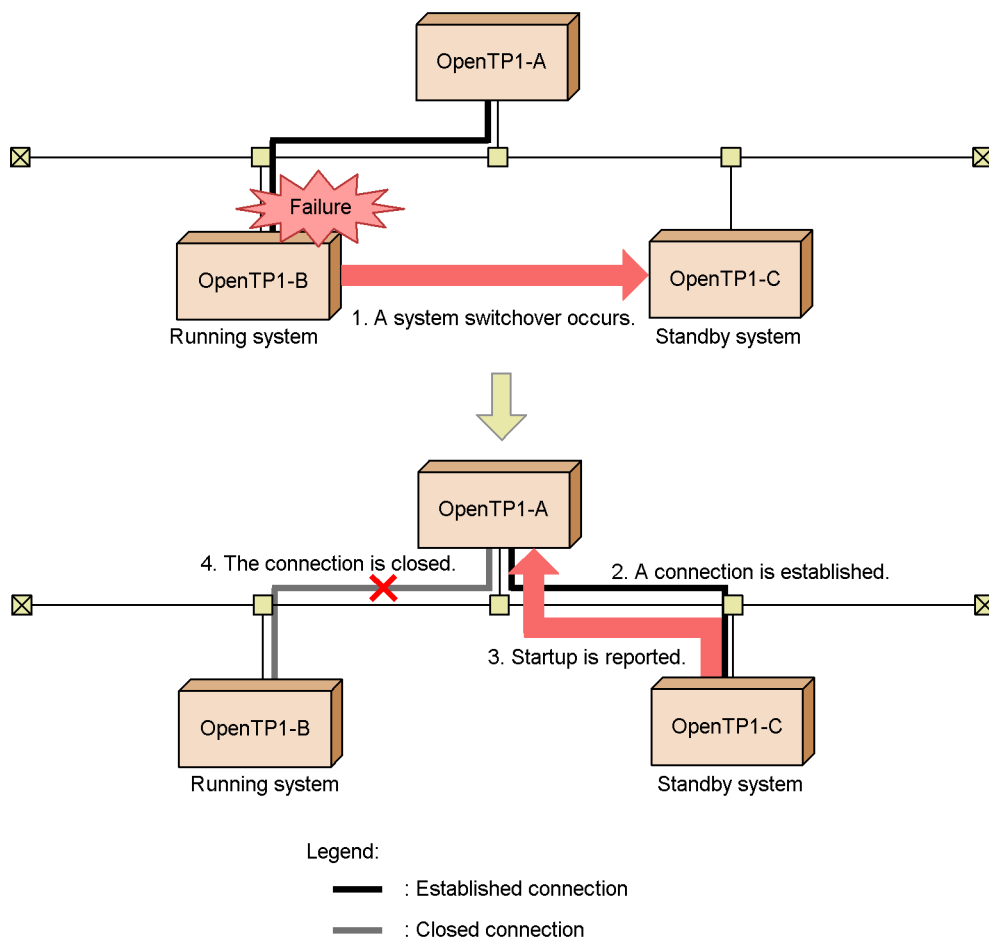
To enable OpenTP1 startup to be notified to another node, specify `Y` in the `name_notify` operand in the system common definition on both the sending and receiving nodes.

OpenTP1 on both nodes must be version 05-02 or later to use this functionality.

Figure 3-22 shows an example of a system configuration when using the startup notification facility at system switchover.



Figure 3-22: Example of using the startup notification facility at system switchover



1. OpenTP1-B goes down due to a server failure or other error, and a system switchover occurs. OpenTP1-A cannot detect the failure in OpenTP1-B, so the connection remains open.
2. The systems are switched, and OpenTP1-C starts on the standby node.
3. If the startup notification facility is enabled, notification that OpenTP1-C has started is sent to OpenTP1-A.
4. OpenTP1-A forcibly closes its connection to OpenTP1-B.

Because communication among OpenTP1-A, OpenTP1-B, and OpenTP1-C resumes in this way from the establishment of a new connection, processing continues without

any communication errors.

If startup fails to be notified to OpenTP1-A for any reason, message KFCA00642-W is output on OpenTP1-C. In this case, you must execute the `namunav1` command on OpenTP1-A. By specifying the `-l` option in the `namunav1` command, you can find out which nodes could not be notified that OpenTP1-C had started.

Note

The startup notification facility cannot be used when multiple instances of OpenTP1 are running on a monitored host, or when multiple instances of OpenTP1 run with the same IP address after a system switchover (an environment with only one LAN board).

## **(2) Node monitoring facility**

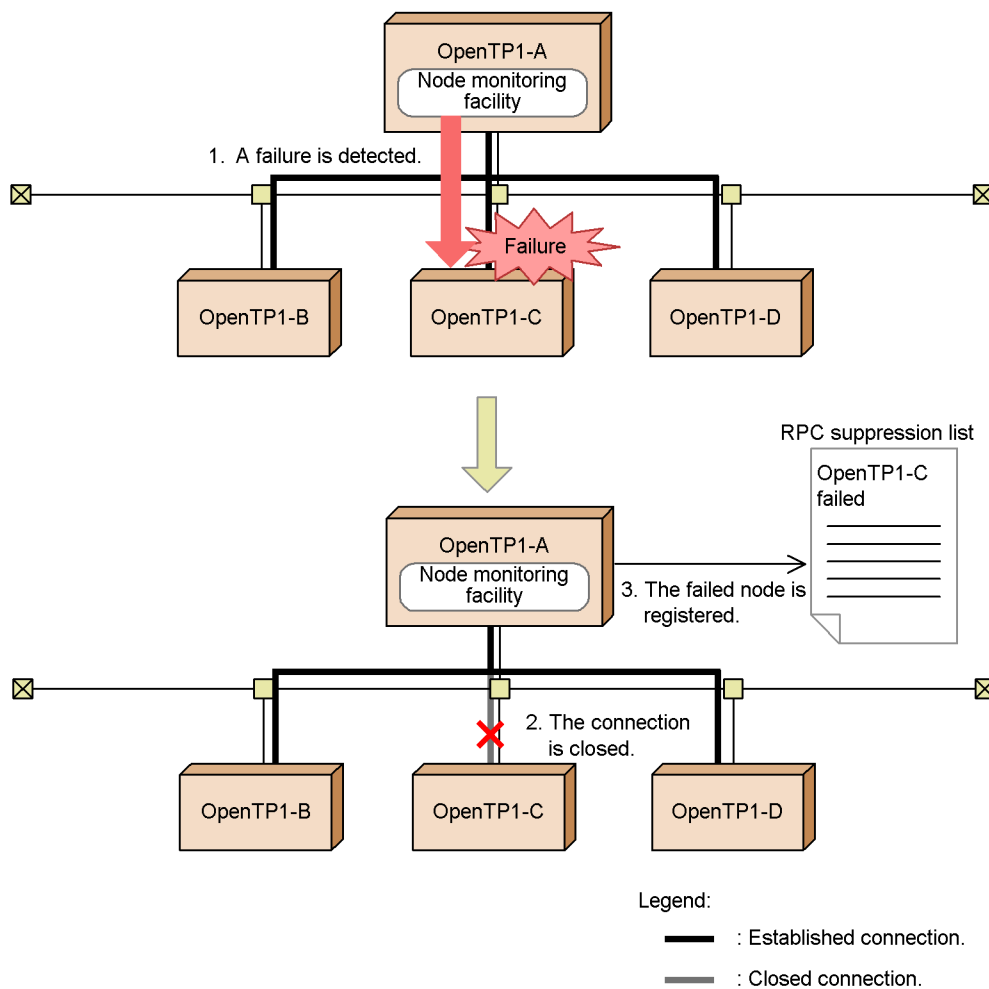
The node monitoring facility polls nodes at regular intervals and detects communication failures.

Using the node monitoring facility, you can monitor the status of OpenTP1 on nodes specified in the `all_node` operand and `all_node_ex` operand in the system common definition. If an OpenTP1 node cannot be detected as active, this facility deletes all cached service information relating to the node and closes the connection.

Node monitoring minimizes errors because failures are detected and failed nodes are forcibly disconnected.

Figure 3-23 shows an example of monitoring other nodes by using the node monitoring facility.

Figure 3-23: Monitoring other nodes by using the node monitoring facility



The node monitoring facility at OpenTP1-A periodically polls OpenTP1-B, OpenTP1-C, and OpenTP1-D.

1. If the OpenTP1-C node goes down, the node monitoring facility detects that OpenTP1-C cannot be reached.
2. OpenTP1-C is disconnected and message KFCA00650-I is output.
3. The failed node is registered in the RPC suppression list<sup>#</sup>. Service information about the node is deleted from the cached service information.

#

An RPC suppression list contains information about nodes on which the OpenTP1 system is inactive.

The node monitoring facility checks whether nodes are active at the intervals specified in the `name_audit_interval` operand of the name service definition. To use the node monitoring facility, specify 1 or 2 in the `name_audit_conf` operand of the name service definition.

The node monitoring facility behaves as follows according to the value specified in the `name_audit_conf` operand:

■ 1 specified in `name_audit_conf`

Send-only nodes are monitored. When send processing ends, the node monitoring facility behaves as follows:

- If send processing by a previously active node fails

The monitored node is judged to be in the stopped state. Message KFCA00650-I is output to the standard output, and information about the node is entered in the RPC suppression list.

- If send processing by a previously stopped node succeeds

The previously stopped node is judged to be in an active state. Message KFCA00651-I is output to the standard output, and information about the node is deleted from the RPC suppression list.

■ 2 specified in `name_audit_conf`

Send/receive nodes are monitored. When send/receive processing ends, the node monitoring facility behaves as follows:

- If send or receive processing by a previously active node fails<sup>#</sup>

The monitored node is judged to be in the stopped state. Message KFCA00650-I is output to the standard output, and information about the node is entered in the RPC suppression list.

- If send or receive processing by a previously stopped node succeeds

The previously stopped node is judged to be in an active state. Message KFCA00651-I is output to the standard output, and information about the node is deleted from the RPC suppression list.

#

If a timeout occurs, receive processing is assumed to have failed. The timeout value is the value specified in the `name_audit_watch_time` operand of the name service definition.

While the node monitoring facility is enabled, the following action occurs according to the status of the monitored node.

- If communication with a previously active node fails  
The node with which communication failed is judged to be in the stopped state. Message KFCA00650-I is output to the standard output, and information about the node is entered in the RPC suppression list.
- If communication with a previously stopped node succeeds  
The previously stopped node is judged to be in an active state. Message KFCA00651-I is output to the standard output, and information about the node is deleted from the RPC suppression list.

The `namalivechk` command is another way of checking whether nodes are active. Table 3-4 describes the differences between using the node monitoring facility and the `namalivechk` command.

*Table 3-4:* Comparison of node monitoring using the node monitoring facility and the `namalivechk` command

Item	Node monitoring facility	namalivechk command
Monitored nodes	<ul style="list-style-type: none"> <li>• All nodes specified in the <code>all_node</code> operand of the system common definition (whether active or not)</li> <li>• All nodes specified in the <code>all_node_ex</code> operand of the system common definition (whether active or not)</li> </ul>	<ul style="list-style-type: none"> <li>• Nodes specified in the <code>all_node</code> operand of the system common definition, on which OpenTP1 has not been detected as inactive</li> <li>• All nodes specified in the <code>all_node_ex</code> operand of the system common definition (whether active or not)</li> </ul>
Operation when an inactive node is detected	<ul style="list-style-type: none"> <li>• If the node is specified in the <code>all_node</code> operand, information about the node is entered in the RPC suppression list if not already entered. If the node has already been entered, no action is taken.</li> <li>• The connection with the inactive node is closed.</li> <li>• Cached service information about the inactive node is deleted.</li> </ul>	<ul style="list-style-type: none"> <li>• Information about any node specified in the <code>all_node</code> operand that is found to be inactive is entered in the RPC suppression list.</li> <li>• The connection with the inactive node is closed.</li> <li>• Cached service information about the inactive node is deleted.</li> </ul>
Operation when an active node is detected	If the node is specified in the <code>all_node</code> operand and has been entered in the RPC suppression list, information about the node is deleted from the list.	No action

#### Note

- The node monitoring facility cannot be used when multiple instances of OpenTP1 are running on a monitored host, or when multiple instances of OpenTP1 run with the same IP address after a system switchover (an

environment with only one LAN board).

- Change the following operands to tune the sensitivity with which a node-down condition is detected by the node monitoring facility:
  - If 1 is specified in the `name_audit_conf` operand:
    - Change the `ipc_conn_interval` operand in the system common definition.
    - If 2 is specified in the `name_audit_conf` operand:
      - Change the `name_audit_watch_timeg`
  - A maximum of 60 nodes can be monitored concurrently by the node monitoring facility. If more than 60 nodes are specified in the `all_node` and `all_node_ex` operands in the system common definition, monitoring is performed for each group of 60 or less in turn.
  - If a large number of nodes are specified in the `all_node` and `all_node_ex` operands in the system common definition, use of the node monitoring facility may affect the RPCs issued by UAPs. In this case, the value specified in the `name_audit_interval` operand should not be too small. Also, do not use the `namalivechk` command too soon after the previous execution.

### **(3) Facility for monitoring nodes registered in the RPC suppression list**

The name service can check at 180-second intervals whether nodes registered in the RPC suppression list are active again. This facility is separate from the node monitoring facility. Specify whether to use this facility in the `name_rpc_control_list` operand of the name service definition.

Decide whether to use the facility for monitoring nodes registered in the RPC suppression list according to the setting for the node monitoring facility. For example, monitoring of nodes registered in the RPC suppression list should be disabled if either of the following occur:

- A node that has been restored after a failure may be deleted from the RPC suppression list, even though the time specified in the `name_audit_interval` operand has not yet elapsed. If this occurs, message KFCA00651-I will not be output.
- If 2 is specified in the `name_audit_conf` operand, message KFCA00650-I may be output periodically.

If the facility for monitoring nodes registered in the RPC suppression list is disabled and a value of 180 seconds or longer is specified in the `name_audit_interval` operand, it takes longer than usual for a recovered node to be deleted from the RPC suppression list.

We recommend that you set the node monitoring facility and the facility for monitoring

nodes registered in the RPC suppression list as follows:

- If the `name_audit_conf` operand is set to 1 or 2, and a value of less than 180 is specified in the `name_audit_interval` operand

We recommend that you specify `N` in the `name_rpc_control_list` operand.

- If the `name_audit_conf` operand is set to 0 or omitted

We recommend that you specify `Y` or omit the `name_rpc_control_list` operand.

Specifying 0 or omitting the `name_audit_conf` operand, and specifying `N` in the `name_rpc_control_list` operand, disables both the node monitoring facility and the facility for monitoring nodes registered in the RPC suppression list. This results in the following situation:

- A registered node cannot be deleted from the RPC suppression list unless there is communication between that node and the local node.
- If the local node is not specified in the `all_node` operand of a node registered in the RPC suppression list, that node cannot be deleted from the RPC suppression list unless OpenTP1 is restarted on the local node.

#### **(4) Node information display**

By executing the `namsvinf` command, you can view the IP address, activity status, and the port number of the name service for OpenTP1 nodes. This information can be displayed for OpenTP1 nodes specified in the `all_node` operand and `all_node_ex` operand in the system common definition.

### **3.2.4 Communication via RPCs that use the XATMI interface**

The XATMI interface is an application programming interface (API) that uses client/server communication and conforms to the DTP model defined in X/Open. UAP processes in an OpenTP1 system can communicate with each other using this interface.

Either TCP/IP or OSI TP can be used as the communication protocol with the XATMI interface.

An SUP and SPP can use the XATMI interface to communicate with each other. A stub created from the XATMI interface definition file must be linked to both the SUP and SPP.

XATMI interface functions cannot be used when creating an MHP.

For details about communication using the XATMI interface, see the *OpenTP1 Programming Guide*.

**(1) XATMI communication modes**

The XATMI interface provides the following modes of communication:

- Request/response service paradigm

Communication in which a request is sent to a service function and a response is received. This mode of communication requests a service and receives the result, in the same manner as remote procedure calls in OpenTP1.

- Conversational service paradigm

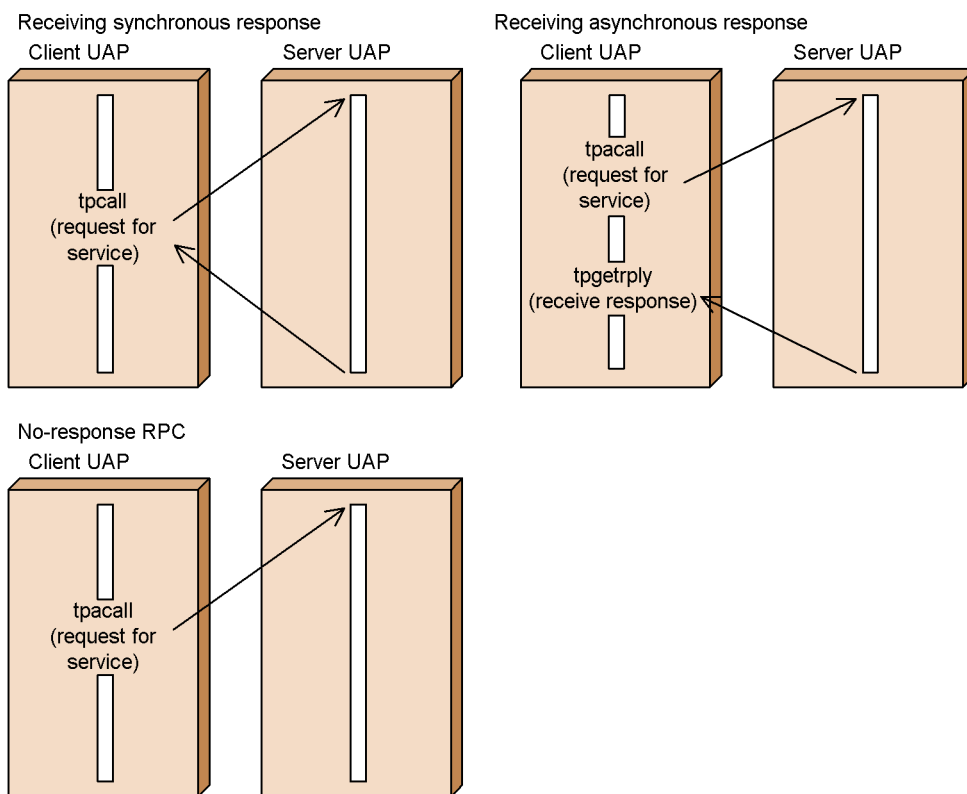
Communication in which data is exchanged via a connection established with a service function started on another node. Interactive services cannot be used when the communication protocol is OSI TP.

Figure 3-24 provides an overview of communication using the XATMI interface.

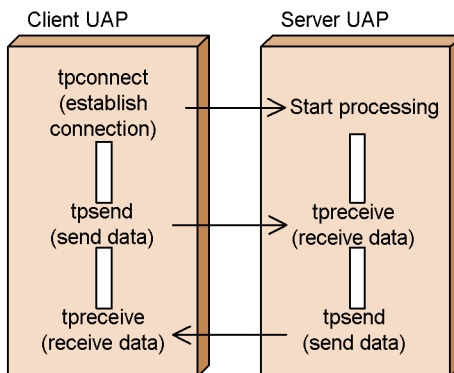


Figure 3-24: Overview of communication using the XATMI interface

● Request-response service paradigm



● Conversational service paradigm



**(2) Method of requesting a service**

To request a service, call the required service function in the server UAP, specifying as the argument a name (service name) that identifies the function.

**(3) Data used in communication via the XATMI interface**

When using the XATMI interface, you must set the data type for inter-process communication. The buffer that stores typed data used in communication via the XATMI interface is known as *typed buffer* or *typed record*.

**3.2.5 Communication via RPCs that use the TxRPC interface**

The TxRPC interface is a client/server communication method defined in X/Open. UAP processes in an OpenTP1 system can communicate with each other using this interface.

In TxRPC communication, unlike other client/server modes of communication, the client calls a user-created function to communicate with another UAP.

A UAP that uses the TxRPC interface can also communicate with another UAP by using OpenTP1 remote procedure calls (`dc_rpc_call` function).

**(1) TxRPC communication modes**

There are two modes of TxRPC communication: using DCE RPCs or not using DCE RPCs.

- IDL-only TxRPC

A method of creating UAPs using only the files that are created from the IDL compiler. DCE is not a prerequisite for IDL-only TxRPC.

- RPC TxRPC

A method of using a DCE RPC as the communication protocol. This version does not support RPC TxRPC.

For details about the TxRPC communication, see the *OpenTP1 Programming Guide*.

**(2) Application types that can communicate via TxRPC**

The following UAPs can be set up so they communicate using TxRPC:

- Client UAP (SUP)
- Server UAP (SPP)

For TxRPC communication, the library requirements differ depending on the type of UAP that is created.

**(a) Creating an SUP or SPP**

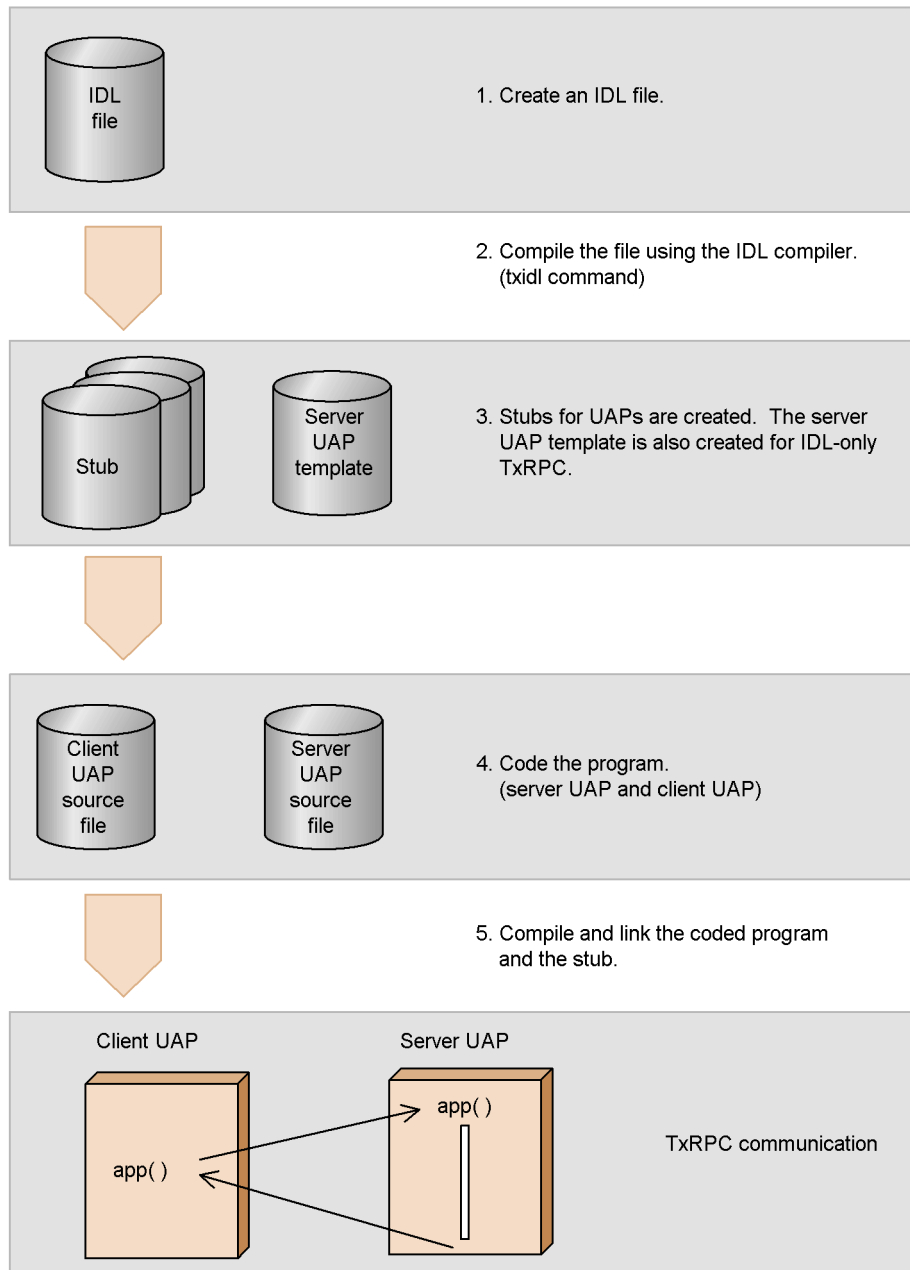
The following product must be included in the system:

- TP1/Server Base

**(3) Creating a UAP for the TxRPC communication**

Figure 3-25 illustrates how to create a UAP that communicates using TxRPC.

Figure 3-25: Overview of TxRPC communication



For details about how to write a UAP that communicates using TxRPC, see the manuals *OpenTPI Programming Guide* and *OpenTPI Programming Reference C*

*Language.*

---

## 3.3 Message Control

---

An OpenTP1 system can communicate with a non-OpenTP1 system by using message control facilities (MCF) that comply with a variety of communication protocols. Using MCF, you can configure a parallel or vertically distributed network system that includes non-OpenTP1 systems.

The following software products are prerequisites for using the OpenTP1 message control facilities:

- TP1/Message Control (for managing message control)
- TP1/NET/Library (for controlling the network)
- TP1/NET/xxxx (for controlling the interface of a specific communication protocol)

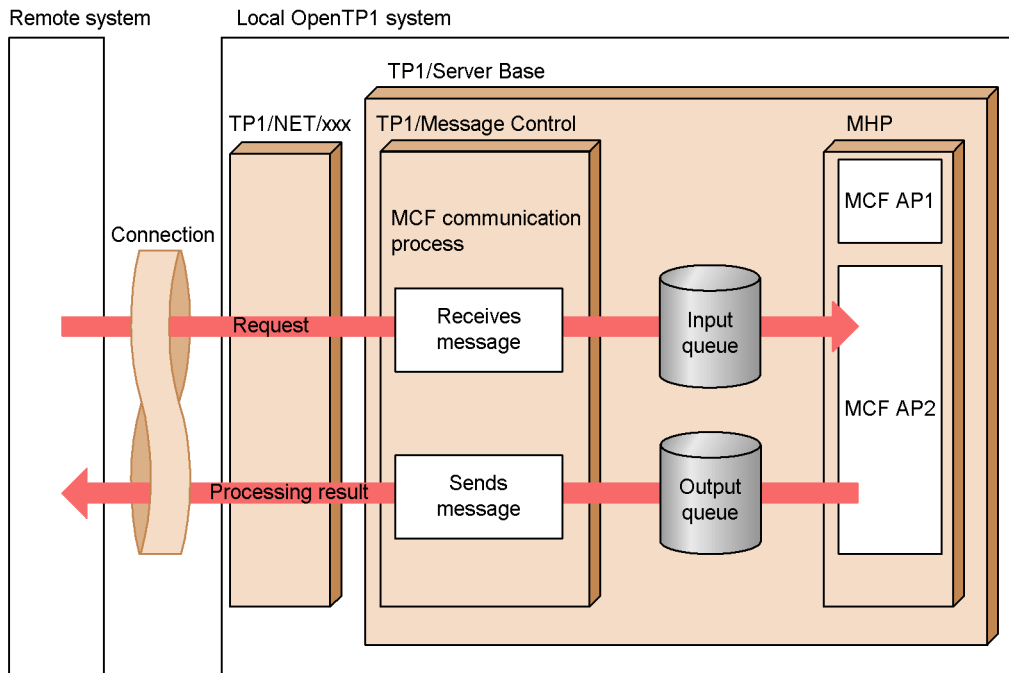
UAPs that use message control are known as message-handling programs (MHPs). In addition, some messages can be sent and received using an SPP.

### 3.3.1 Overview of sending and receiving messages using MCF

Messages are commonly used for communicating between an OpenTP1 and a non-OpenTP1 system. An OpenTP1 system that can send or receive messages in real time is said to have an *MCF message-exchange configuration*. The message control facilities, which are usually abbreviated to MCF, manage an MCF message-exchange system. The message control facilities are provided by the software products TP1/Message Control, TP1/NET/Library, and the products that correspond to the relevant communication protocol.

Figure 3-26 illustrates the above example of MCF message exchange.

Figure 3-26: Overview of sending and receiving a message by using MCF

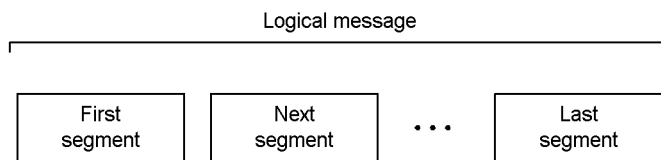


### 3.3.2 Message structure

A meaningful unit of communication is known as a *logical message*. A message unit handled by a UAP function is known as a *segment*. A logical message can consist of one or more segments.

Figure 3-27 shows the relationship between a logical message and its segments.

Figure 3-27: Segments in a logical message



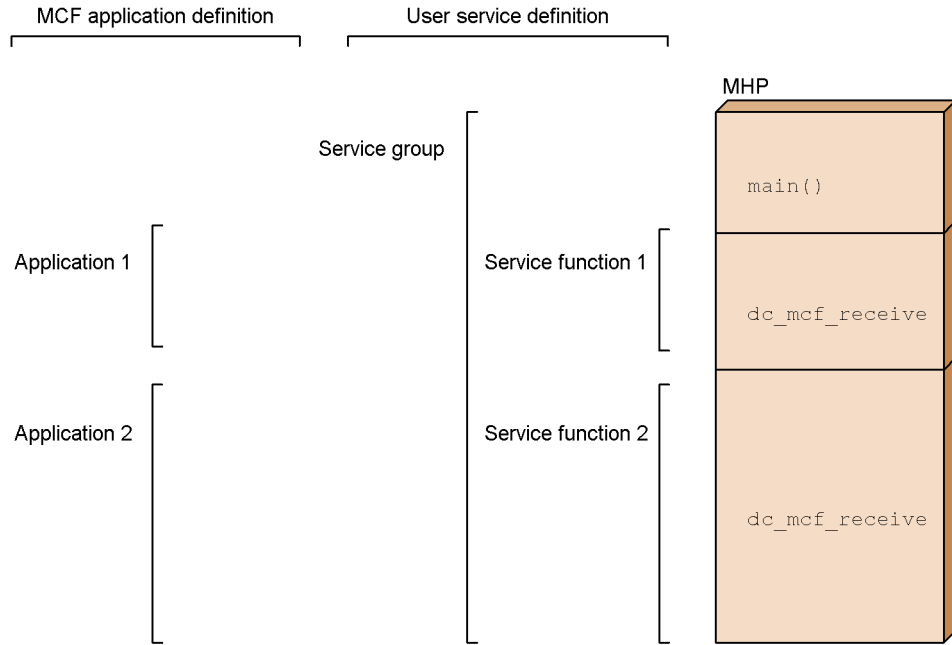
### 3.3.3 Application program structure and application name

An MHP consists of a main function and service functions. To register an MHP executable file with TP1/Server Base, you must create a *user service definition*. You must also specify the MHP in an *MCF application definition* so that the MHP can be managed by MCF. These definition procedures associate the MHP service functions

with an application name, enabling the service functions to be used as an MCF application.

Figure 3-28 shows the relationship between the application program structure and application name.

*Figure 3-28: Relationship between application program structure and application name*



### 3.3.4 Synchronous and asynchronous communication functions, and messages

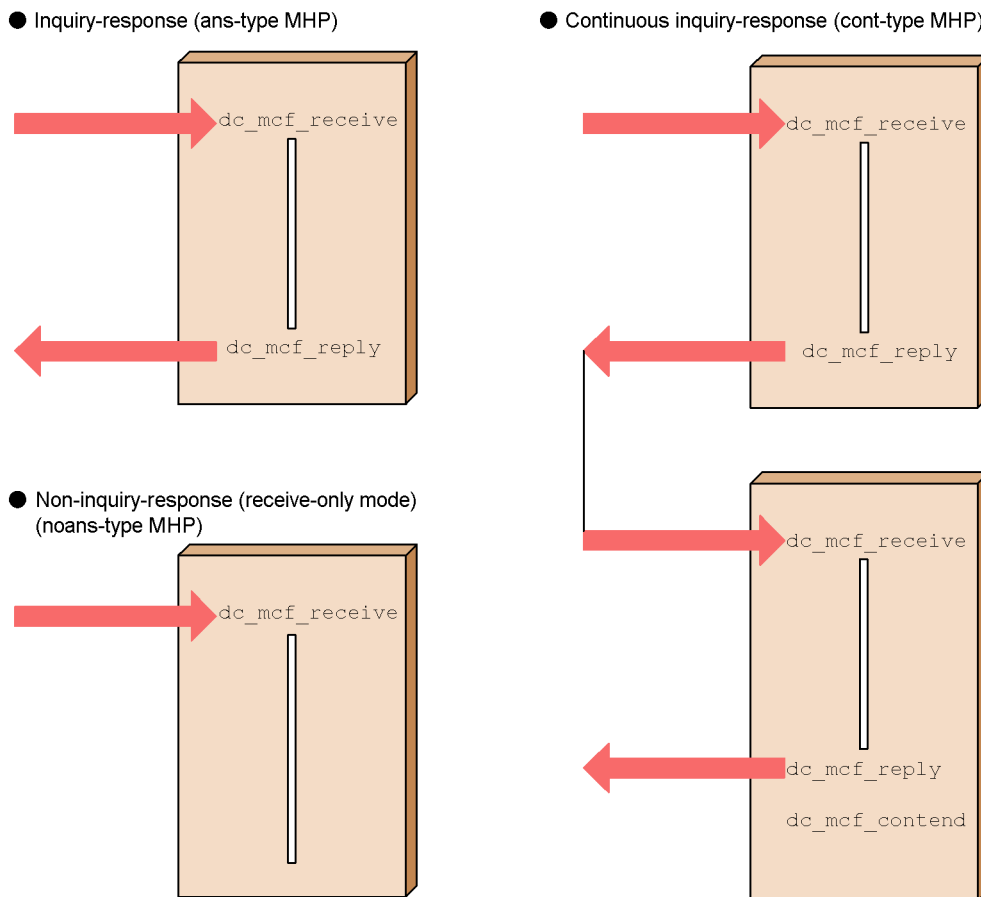
The following describes the message communication process when using an MHP. The process differs according to the communication protocol.

The message processing described below applies only to an MHP.

Figure 3-29 shows message processes and the type of MHP application.



Figure 3-29: Message processing and MHP application type



### (1) Inquiry-response communication

In inquiry-response communication, a message is received from another system, and a response message is sent back. The `dc_mcf_receive` function is used to receive a message, and the `dc_mcf_reply` function is used to send the response.

An MHP that performs communication in inquiry-response mode is specified as a *response-type (ans-type)* application. If a response-type MHP ends without calling the `dc_mcf_reply` function, an error is assumed and the MHP ends abnormally.

### (2) Non-inquiry-response communication (receive-only mode)

In non-inquiry-response communication, a message is received from another system, but no response is returned. The `dc_mcf_receive` function is used to receive the message.

An MHP that performs non-inquiry-response processing (receive-only mode) is specified as a *non-response (noans-type)* application.

### (3) Continuous inquiry-response communication

Continuous inquiry-response communication consists of a series of inquiry-response messages. The `dc_mcf_receive` function is used to receive a message from another system, and the `dc_mcf_reply` function is used to send the response. Subsequent messages are received after each response, either by the same MHP or by a different MHP. Continuous inquiry-response communication is ended by issuing the `dc_mcf_contend` function.

An MHP that performs continuous inquiry-response processing is specified as a *continuous inquiry response type (cont-type)* application.

## 3.3.5 Messages independent of the above communication modes

The following describes the types of messages that can be sent and received by an MHP or SPP.

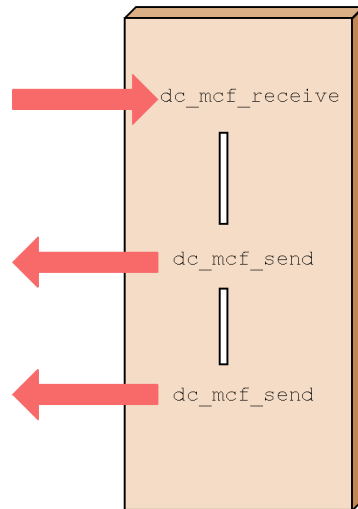
### (1) Send-only messages

Messages that are sent by an OpenTP1 UAP. The `dc_mcf_send` function is used to transmit a send-only message. The message is sent to the other system when the UAP transaction that called the `dc_mcf_send` function has been committed.

Figure 3-30 illustrates send-only messages.

Figure 3-30: Overview of send-only messages

#### ● Send-only messages



**(2) Resending messages**

A message already sent from an OpenTP1 UAP can be resent. The `dc_mcf_resend` function is used to resend a message.

**(3) Synchronous communication functions and messages**

In addition to asynchronous communication functions, OpenTP1 supplies *synchronous communication functions* that are executed regardless of the transaction determination. Issuing a synchronous communication function from a UAP allows the UAP to communicate with another system without waiting for a transaction determination. In such a case communication resembles interactive processing.

**(a) dc\_mcf\_sendsync (synchronous send function)**

The `dc_mcf_sendsync` function is used to send a message synchronously. When the function is called from a UAP, the UAP processing stops while the message is written to the output queue. Once the message has been sent to the other system, the function returns.

**(b) dc\_mcf\_recvsync (synchronous receive function)**

The `dc_mcf_recvsync` function is used to receive a message synchronously. When the function is called from a UAP, the UAP processing stops while a message directed to the application specified in the function is located in the input queue. If such a message is found, the segment is received and the function returns. If no such message is found, the UAP waits for one to arrive.

**(c) dc\_mcf\_sendrecv (synchronous send/receive function)**

The `dc_mcf_sendrecv` function is used to send and receive messages synchronously. When the function is called from a UAP, the UAP processing stops while the message is written to the output queue. The message is sent to the other system, a reply is received, and then the function returns.

**3.3.6 Message-control transactions**

The following describes the processing of message-control transactions.

**(1) MCF message processed as an MHP transaction**

On receipt of an MCF message from another system, the MHP service is called and an MHP transaction begins. If the MHP that processed the message ends normally, the transaction is committed and the MHP process takes effect.

An MHP can either commit or roll back a transaction. To commit an MHP transaction, use the `dc_mcf_commit` function. To roll back an MHP transaction, use the `dc_mcf_rollback` function.

**(2) MCF message not processed as an MHP transaction**

You can choose not to handle a received MCF message as an MHP transaction. This

makes processing more efficient, although the message cannot be recovered if an error occurs in the UAP. An MHP that does not process MCF messages as transactions is known as *non-transaction attribute MHP*. To use this type of MHP, specify `nontrn` in the `trnmode` operand of the application attribute definition.

### 3.3.7 Starting user application programs

Programs in a system that sends and receives MCF messages do not need to be constantly running. For example, MCF can start an MHP whenever a message is received that needs to be processed by that MHP; or a programmer can write a program that starts whenever an MCF error event occurs. This section describes how programs can be started in a MCF message-exchange system. An MHP or SPP can be started:

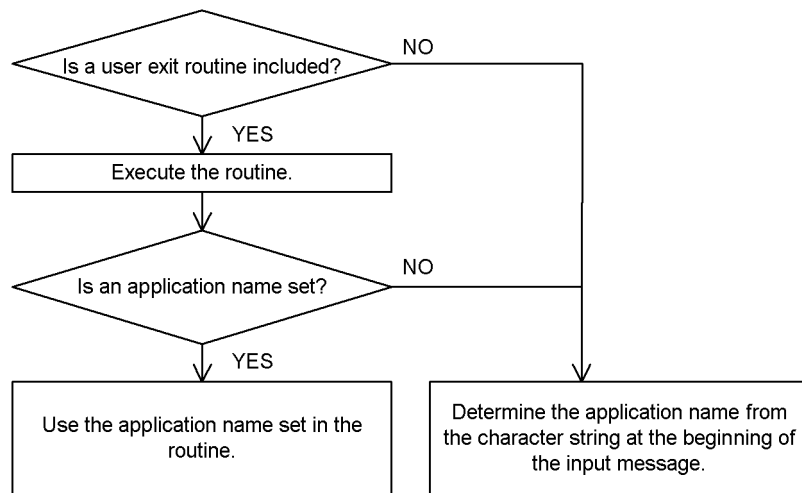
- by MCF when MCF receives a message from another system
- by the occurrence of an MCF communication event or MCF error event
- by another UAP issuing a function (`dc_mcf_execap()`) to start the MHP or SPP
- by executing the UAP start command `mcfuevt`

#### (1) Starting a user application program on receipt of a message

MCF can determine the application name (that is, what MHP or SPP should be started) from an input message.

Figure 3-31 shows how the application name is determined.

Figure 3-31: Determination of application name



The application name can be determined by including a user exit routine for determining the application name. Alternatively, you can choose not to include such an exit routine.

**(a) When not using an exit routine for determining the application name**

MCF regards the string of eight or fewer characters before the first space in an input message as the application name. OpenTP1 converts the application name into the MHP or SPP service group name and service names, and then starts the MHP or SPP. If the first nine characters are all spaces, the application name is regarded as invalid and an error occurs.

Specify the correspondence between the application name and the service group name and service names in the application attribute definition of the MCF application definition.

**(b) When using an exit routine for determining the application name**

MCF calls the exit routine and regards the name set in the routine as the application name. OpenTP1 converts the application name into the MHP or SPP service group name and service names, and then starts the MHP or SPP.

Specify the correspondence between the application name and the service group name and service names in the application attribute definition of the MCF application definition. Note that even when an exit routine is included in the OpenTP1 system, if the application name is not set in the routine, the string of eight or fewer characters before the first space in the message will be regarded as the application name. If the first nine characters are all spaces, the application name is regarded as invalid and an error occurs.

**(2) Starting a UAP (MHP or SPP) by issuing a function from a UAP**

An MHP or SPP can issue a function (`dc_mcf_execap()`) to start another MHP or SPP. This ability of an MHP or SPP to start another UAP is called the *Application Activate facility*.

Immediately after a normal termination of the MHP or SPP that requested the start or after a specified time has elapsed, the *application startup process* starts the specified MHP or SPP.

If an MHP issues the function `dc_mcf_rollback()` to roll back a transaction and `retry` is specified in the function parameters, the rolled-back MHP can be restarted and processing can be re-executed.

For a UAP to issue the function `dc_mcf_execap()` or `dc_mcf_rollback()` and start or restart an application as described above, the *application startup process* must be started. To enable this, create the MCF communication configuration definition for the *application startup process*. By creating this definition, you can reduce the load on the MCF communication process because the *application startup process* performs processing that does not depend on the protocol.

The inter-OpenTP1 logical channel that the MHP or SPP uses to pass data to MCF when requesting the start of another UAP is called the *internal channel*. The internal channel is defined in the *application startup environment definition*. MCF also uses

the internal channel to pass data to the application program that is to be started.

**(3) Starting a UAP (MHP) by an MCF event**

OpenTP1 administrators should be aware of certain events, such as failures or establishing or releasing a connection, that sometimes occur during online processing. OpenTP1 reports such events in messages produced by *MCF events*. A programmer can create an MHP that is started by such an MCF event.

**(a) MCF events**

- MCF error events
- MCF communication events

MCF error events are events such as errors or failures. MCF communication events are events such as the establishing or releasing of connections.

The two types of events are identified by MCF event codes. In the MCF application definition, you can specify the correspondence between MCF event codes and MCF applications.

The application startup process or the MCF communication process starts the service of the MHP that corresponds to the MCF event code. The application startup process or MCF communication process starts the MHP for an MCF error event. The MCF communication process starts the MHP that processes an MCF communication event. When a programmer develops an MHP that processes an MCF error event, the programmer must also write an MCF communication configuration definition for the application startup process.

You can issue an error event to report an error in a communication event. Use the `errvnt` operand of the application attribute definition to specify whether or not to enable this report. However, you cannot issue an error event to report an error in an error event.

The following subsections list the MCF error events, their causes, and some examples of what kind of actions a program should take when the event occurs. In addition to these error events, there are also communication events for individual protocols: for events such as establishing or releasing connections. For details on MCF events, see the applicable *OpenTP1 Protocol* manual.

Table 3-5: List of MCF events

MCF event name	MCF event code	Reason for the MCF event notification	Example of processing executed by an MHP started by an MCF event
MCF event that reports detection of an invalid application name	ERREVT1	This MCF event occurs when the MCF application definition does not contain the MCF-application name of the message.	A programmer could write an MHP for this MCF event so that the MHP notifies the user that the appropriate MCF-application name was absent, and outputs the corresponding response message for inquiry messages.
MCF event that reports discarding of a message	ERREVT2	<p>This MCF event occurs when the message that MCF received from the input queue or the message placed in the input queue by the facility for immediately starting the application is discarded because:</p> <ul style="list-style-type: none"> <li>• An error occurred in the input queue.</li> <li>• The service, service group, and MCF-application of the MHP are shut down.</li> <li>• The service, service group, and MCF-application of the MHP have secure status.</li> <li>• The MHP terminated abnormally before the segment was passed to the MHP segment-receiving function.</li> <li>• There was no MHP service corresponding to the MCF-application name.</li> <li>• MCF cannot start the SPP.</li> <li>• The MHP is not started.</li> </ul>	A programmer could write an MHP for this MCF event so that the MHP notifies the user that the message was discarded, and outputs the corresponding response message for inquiry messages.

### 3. Functions

MCF event name	MCF event code	Reason for the MCF event notification	Example of processing executed by an MHP started by an MCF event
MCF event that reports UAP abnormal termination	ERREVT3	This event occurs when an MHP terminates abnormally after the segment has been passed to the <code>dc_mcf_receive</code> function that is called in the MHP, or when the MHP rolls back. <sup>#</sup>	A programmer could write an MHP for this MCF event so that the MHP notifies the user that the UAP terminated abnormally or rolled back. If the message received is an inquiry message, a response message can be sent.
MCF event that reports discarding of a timer-start message	ERREVT4	This event occurs when the message is discarded that was placed in the input queue by the facility for starting the application with the timer because of the reason indicated by ERREVT2.	A programmer could write an MHP for this MCF event so that the MHP notifies the user that a message to request the timer start was discarded, and outputs the corresponding response message for inquiry messages.
MCF event that reports discarding of an unprocessed message	ERREVT4	This MCF event occurs when an unprocessed message to be sent to a remote system is discarded because: <ul style="list-style-type: none"> <li>• Excessive time (a timeout) is taken to process the message during normal termination of an MCF.</li> <li>• The output queue was deleted by the <code>mcf_tdlqle</code> command or the <code>dc_mcf_tdlqle</code> function.</li> <li>• The <code>dcstop</code> command was executed while there was an unprocessed timer start request.</li> </ul>	A programmer could write an MHP for this MCF event so that the MHP notifies the user that an unprocessed send message was discarded.
MCF event that reports a send error	SERREVT	This MCF event occurs when a protocol error occurs during message transmission.	A programmer could write an MHP for this MCF event so that the MHP notifies the user that the message could not be sent due to a protocol error.



MCF event name	MCF event code	Reason for the MCF event notification	Example of processing executed by an MHP started by an MCF event
MCF event that reports completion of send processing	SCMPEVT	This MCF event occurs when a message is successfully sent to the other system.	A programmer could write an MHP for this MCF event so that the MHP notifies the user that the message was successfully sent to the other system.
MCF event that reports an error	CERREVT(V ERREVT)	This MCF event occurs when a connection error or logical terminal error occurs in the communication management program. No error is reported when a connection retry is defined in the program.	A programmer could write an MHP for this MCF event so that the MHP notifies the user that an error occurred in the connection or logical terminal.
MCF event that reports the connection status	COPNEVT(V OPNEVT)	This MCF event occurs when a connection is established.	A programmer could write an MHP for this MCF event so that the MHP notifies the user that messages can be sent and received.
	CCLSEVT(V CLSEVT)	This MCF event occurs when a connection is successfully released.	A programmer could write an MHP for this MCF event so that the MHP notifies the user that messages can no longer be sent and received.

#### Note

ERREVT1, ERREVT2, ERREVT3, ERREVT4, and ERREVTa indicate error events.

SERREVT, SCMPEVT, CERREVT, COPNEVT, and CCLSEVT indicate communication events.

#

Exceptions are when *r* is specified in the *recvmsg* operand of the MCF application definition (*mcfaalcap -g*) and when *DCMCFRTRY* or *DCMCFRRTN* is specified in *action* of *dc\_mcf\_rollback*.

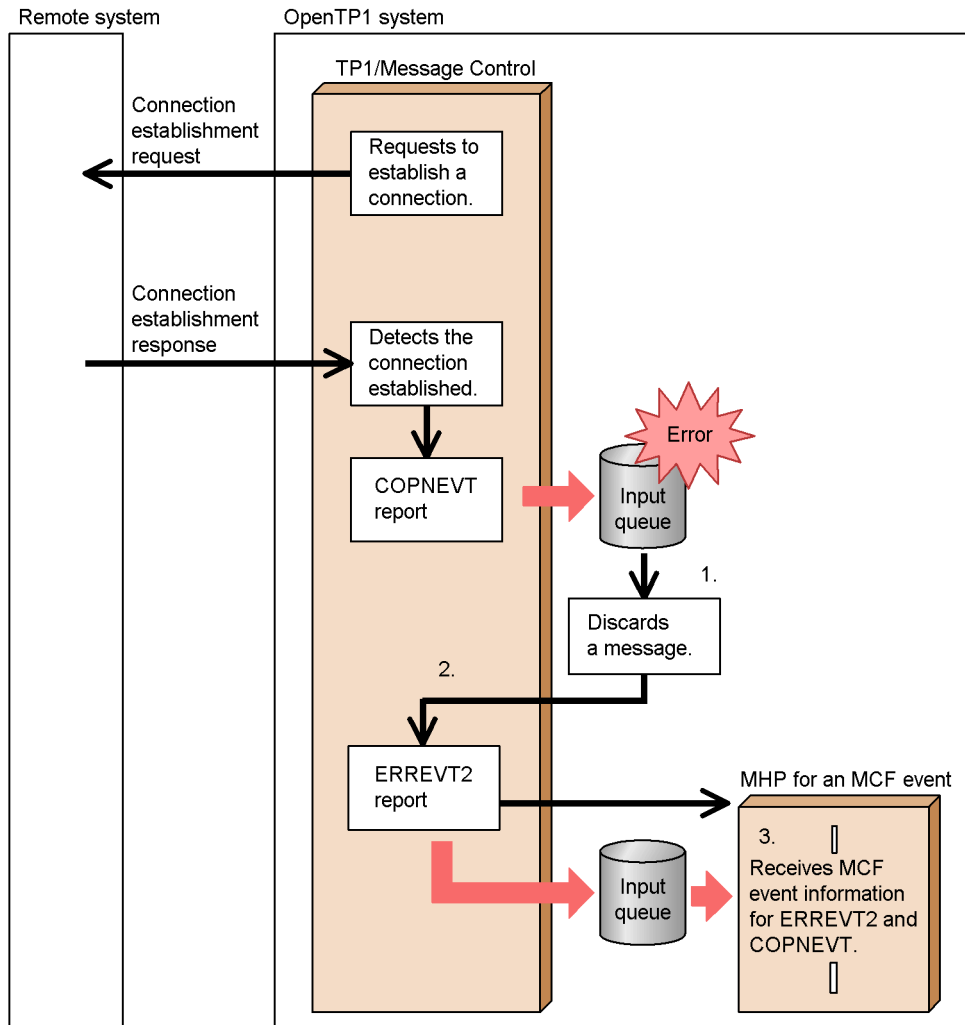
**(b) Reporting a communication event error**

The facility for reporting a communication event error to the user by an error event is available. This facility allows the user to use an MHP for an MCF event to recover the failed communication event. A communication event message can be received at the second segment.

To use the facility for reporting a communication event error, specify *yes* in the *errevt* operand of the *-n* option in the application attribute definition.

Figure 3-32 shows an overview of ERREVT2 report for a COPNEVT error.

*Figure 3-32: Overview of ERREVT2 report for a COPNEVT error*



1. COPNEVT is discarded due to an error in the input queue.
2. Control returns to the MCF. ERREVT2 is reported, and an MHP started by an MCF event is scheduled for ERREVT2.
3. MCF event information for ERREVT2 can be received in the first segment. MCF event information for the discarded COPNEVT can be received in the second segment. The user can deal with the discarded message by referencing the received MCF event information and executing the processing that should have been executed in the MHP for event COPNEVT in the MHP for event ERREVT2.

#### **(4) Starting a UAP by using a command**

A user can use the command `mcfuevt` to start an MHP or SPP. Even for MHPs that start when a message is received, using `mcfuevt` immediately to start an MHP enables a message to be sent to another system.

Only `noans` MHPs can be started with `mcfuevt`, so specify `noans` in the MCF application definition of the MHP that is to be started by `mcfuevt`.

##### **(a) Defining a UAP started by a command**

For a program to be started by `mcfuevt`, you must specify various operands in the MCF application definition `mcfaalcap`. In the `-n` operand of `mcfaalcap`, specify:

```
name=UCMDEVT
kind=user (or omit)
type=noans (or omit)
```

##### **(b) Starting a UAP**

Execute the command `mcfuevt` to start an MHP or SPP. As parameters of the `mcfuevt` command, specify the MCF communications process identifier, and the input message to be passed to the MHP or SPP.

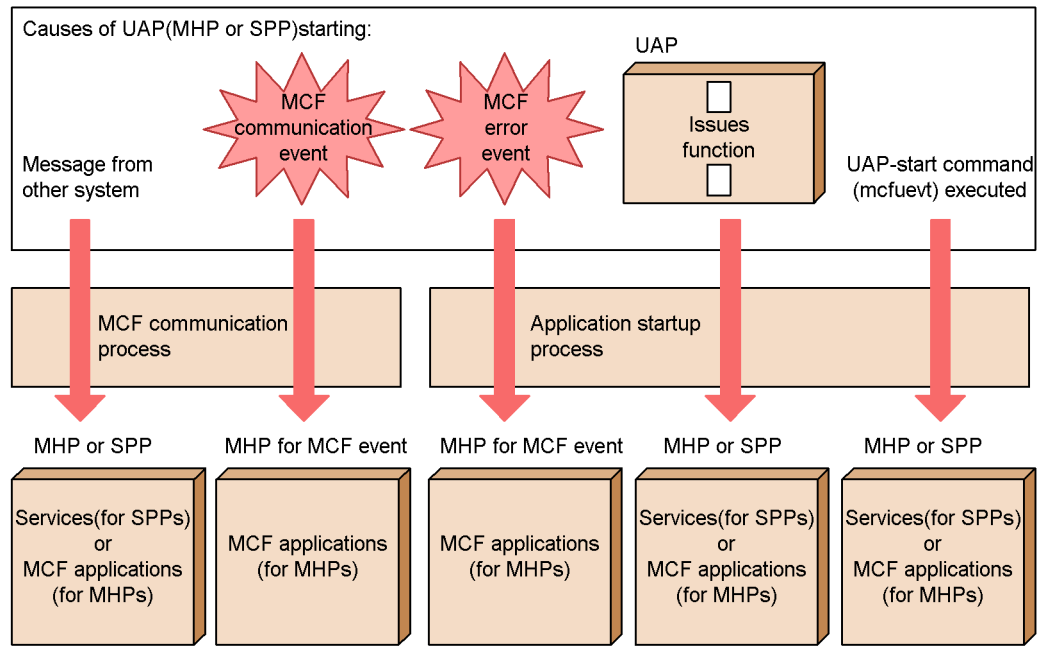
If `UCMDEVT` is not defined and the `mcfuevt` command is executed, the `mcfuevt` command returns an error. In such a case `ERREVT1` does not occur.

An MHP or SPP that is started by a command does not depend on a communications protocol, so it is recommended that you specify the *application startup* process in the MCF communication process that is specified in the `mcfuevt` command.

#### **(5) Causes of MCF starting a UAP**

Figure 3-33 shows the causes of MCF starting a UAP.

Figure 3-33: Causes of MCF starting a UAP



### 3.3.8 MCF message queues and the sending and receiving of messages

#### (1) *Input and output queues*

OpenTP1 uses queues to store messages that have been received or are to be sent.

An *input queue* manages the input messages as processing requests. An *output queue* manages the output messages as processing results.

In the MCF application definition for each application, you can specify whether a disk queue or a memory queue is to be used for the input queue. In the logical terminal definition for each logical terminal, you can specify whether both a disk queue and a memory queue are to be used or only a memory queue is to be used. If you specify use of both queues, send-only messages always use the disk queue. If an inquiry message uses a disk queue, the corresponding response message also uses a disk queue; if an inquiry message uses a memory queue, the corresponding response message also uses a memory queue.

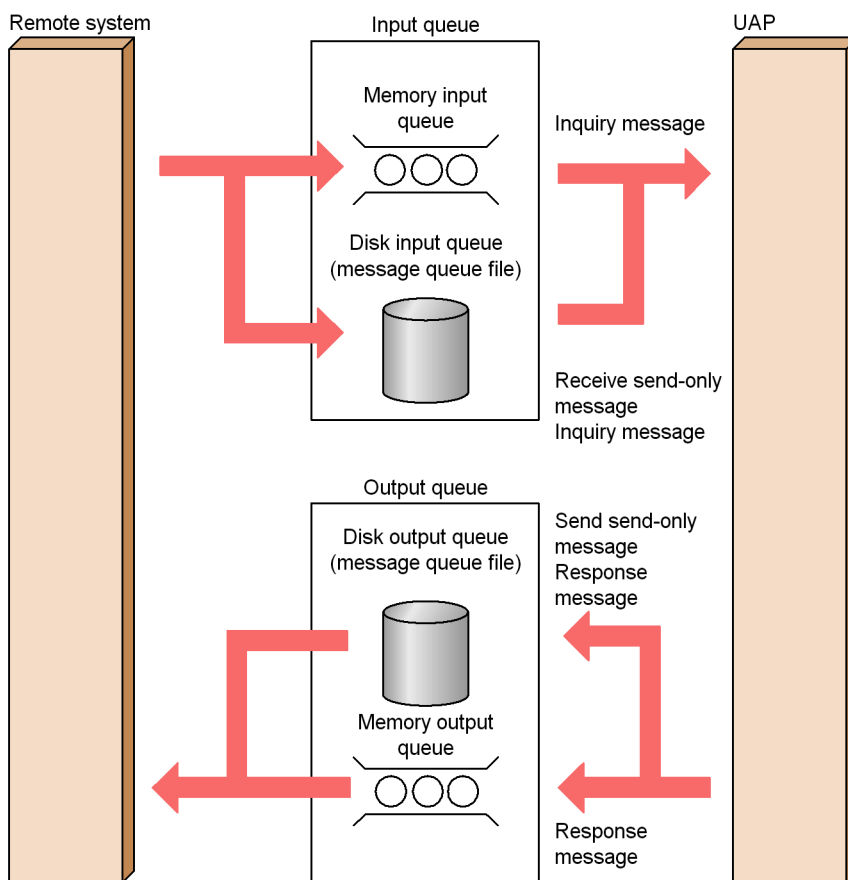
If a disk queue is used, the messages in it can be reused at recovery restart after abnormal termination of the online system. The messages that do not need to be reused at recovery restart should be placed in a memory queue, which provides high-speed processing.

OpenTP1 uses a message queue file as a disk queue. If an OpenTP1 administrator specifies that the disk queue is to be used, the administrator should create an MCF message queue file.

You can place the MCF message queue on hold status by using the `mcftlhldiq` or `mcftlhldoq` command. The hold status is also inherited at restart after abnormal termination of an online system. When the hold status need not be inherited (for example, if an error occurs while replacing a user server using the input queue), the inheritance can be suppressed using an option. When the status inheritance is suppressed it is unnecessary to release the hold status using the `mcftrlsiq` command at restart. For details of the message queue file, see 4.3.1 *Queue files: MCF message queue file*.

Figure 3-34 illustrates message exchange using the input and output queues.

Figure 3-34: Message exchange using the input and output queues



## **(2) Fallback operation using the memory queue instead of the disk queue**

If the disk queue is disabled for some reason at the start of OpenTP1, instead of the disk queue the memory queue can be used to start OpenTP1. This is called a *fallback operation using the memory queue*. In the extended reservation definition for each input queue and output queue, you can specify whether to perform the fallback operation. If you do not perform the fallback operation using the memory queue, the messages to be stored in the disk queue are discarded. In such a case, an MHP designed to process the MCF event that reports discarding of messages (ERREVT2) is started, if it exists.

If the fallback operation using the memory queue is performed, a message (KFCA11065-W or KFCA11066-W) is displayed.

Note that when starting OpenTP1 by the fallback operation using the memory queue, the messages cannot be recovered during complete recovery after shutdown of online processing.

If the disk queue is disabled during online processing, a fallback operation using the memory queue cannot be performed.

For details about the causes and recovery methods for the fallback operation, see the manual *OpenTP1 Operation*.

## **(3) Holding messages in the disk queue**

A disk queue can hold messages even after they are read so that they can be reread. A message held for a possible reread is called a *hold message*. In the MCF message-queue service definition, you can specify the number of messages to be held.

A memory queue cannot hold messages.

For each disk queue, if the ratio of the number of messages waiting to be de-queued and the number of hold messages exceeds a user-defined percentage, a message warning about the usage percentage of the disk queue can be output. In the MCF message-queue service definition, you can specify the disk-queue usage percentage for which a warning message is to be output.

## **(4) Processing when the disk queue is full**

If the disk input queue becomes full, new received messages are discarded. In such a case, if an MHP for ERREVT2 (the MCF event that reports discarding of a message) exists, this MHP is started. If the disk output queue becomes full, the communication functions (`dc_mcf_send()`, `dc_mcf_reply()`, `dc_mcf_sendrecv()`, `dc_mcf_sendsync()`, `dc_mcf_resend()`, `dc_mcf_execap()`) return an error.

### **3.3.9 Message exchange by user application programs**

#### **(1) Receiving messages**

OpenTP1 receives messages from another system through a *connection*. A connection

is a logical channel between OpenTP1 and the other system. As described in 3.3.7 *Starting user application programs*, a message received by OpenTP1 is managed by an *MCF communication process*.

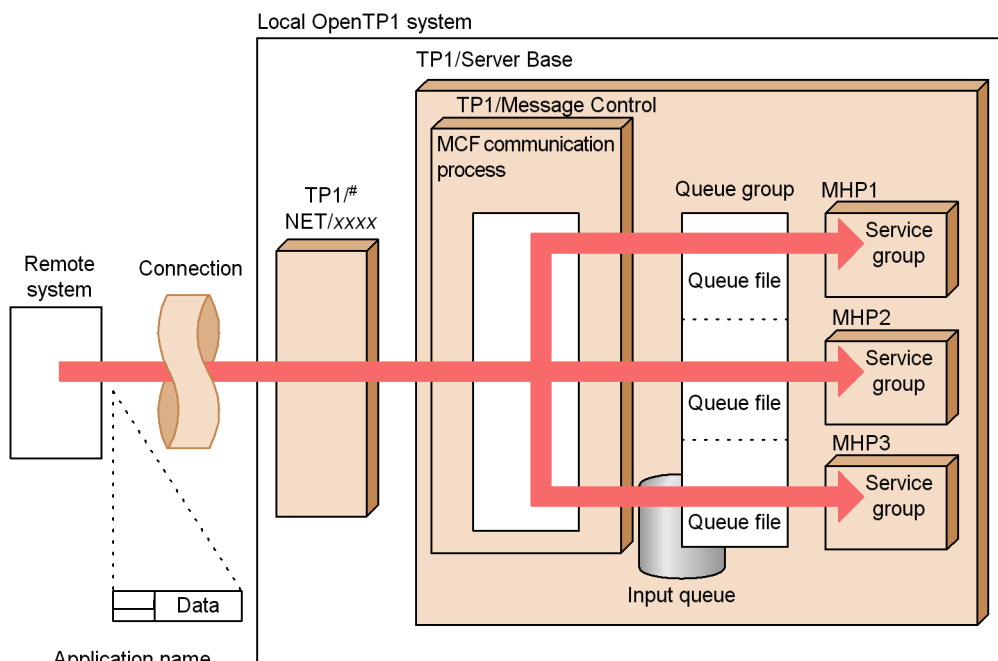
After determining the required SPP or MHP, MCF writes the received message to the input queue for the service group that will process the message. The input queue manages the inputs as service requests. In a disk queue, the part allocated to a service group is called a *MCF message queue file* and the name of the corresponding service group is used as the *MCF message queue file name*.

In the same way as messages for a disk queue, each service group has a separate queue for messages written to a memory queue.

After MCF writes the last segment of a received message to the input queue, MCF records the service request in the schedule queue for the requested MHP. Following the scheduling and starting of an MHP, the MHP uses the segment-receive function `dc_mcf_receive()` to receive the received message in segments.

Figure 3-35 gives an overview of how a UAP receives messages. In the diagram, note that TP1/NET/xxx indicates the product required for the protocol being used.

Figure 3-35: Overview of how a UAP receives messages



#: Product supporting the communication protocol

## **(2) Sending messages**

MCF writes send messages to the output queue according to the *logical terminal name* written in the send function (`dc_mcf_send( )`) parameter. A logical terminal name is a user-specified name assigned to a terminal entity: such as a window of a multi-window workstation, or a printer. OpenTP1 can define either actual terminals or hardware components as *logical terminals* that have logical terminal names. In the logical terminal definition, you can specify logical terminal names.

When the messages to be sent use a disk queue, a disk queue is allocated to each logical terminal. Each of the distributed queues is called an *MCF message queue file* and the name of the corresponding logical terminal is used as the *MCF message queue file name*.

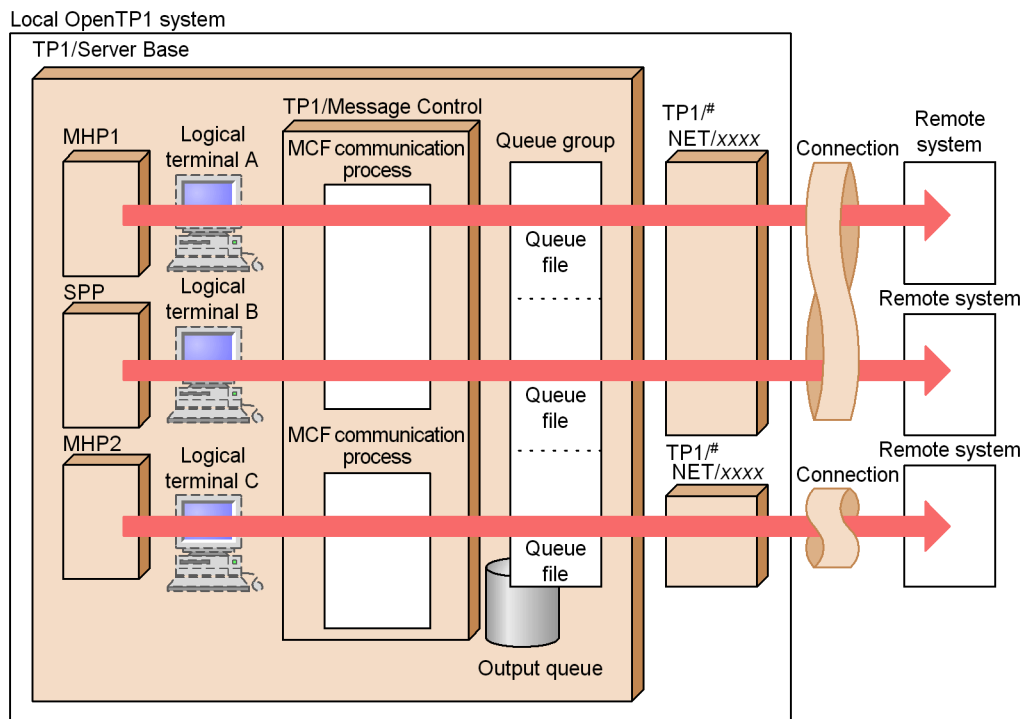
In the same way as messages for a disk queue, each logical terminal has a separate memory queue for messages to be sent.

The MCF communication process created for the protocol sends the message to the other system in segments.

Figure 3-36 gives an overview of how a UAP sends messages. In the diagram, note that TP1/NET/xxx indicates the product required for the protocol being used.



Figure 3-36: Overview of how a UAP sends messages



#: Product supporting the communication protocol

#### (a) Sending a send-only message

The `dc_mcf_send` function can be used in an MHP or SPP to send a send-only message to another system. Specify the logical terminal name as the send destination.

When an MHP or SPP requests that a send-only message be sent, a *message sequential number* is attached according to the message type, and the message is sent to the logical terminal. When sending the first segment, OpenTP1 passes control to the user exit routine for editing message sequential numbers. This exit routine can edit and set message numbers anywhere in the first segment.

#### (b) Sending a response to an inquiry

On receipt of an inquiry message from another system by the `dc_mcf_receive` function, an MHP can send a response message using the `dc_mcf_reply` function to the logical terminal that made the inquiry.

**(c) Sending an inquiry message**

An MHP or SPP can send an inquiry message to another system using the `dc_mcf_send` function.

**(d) Sending a synchronous message**

An MHP or SPP can send a synchronous message to another system using the `dc_mcf_sendsync` function or `dc_mcf_sendrecv` function.

The `dc_mcf_sendsync` function stops UAP processing until the message has been sent to the other system, thus synchronizing with UAP processing by the other system.

The `dc_mcf_sendrecv` function stops UAP processing until a response is received to the message it sent to the other system, thus synchronizing with UAP processing by the other system.

**(3) Starting applications from a UAP**

If you use the Application Activate facility, you can simplify and unify the sending of messages by writing a single MHP that edits and sends messages from several logical terminals.

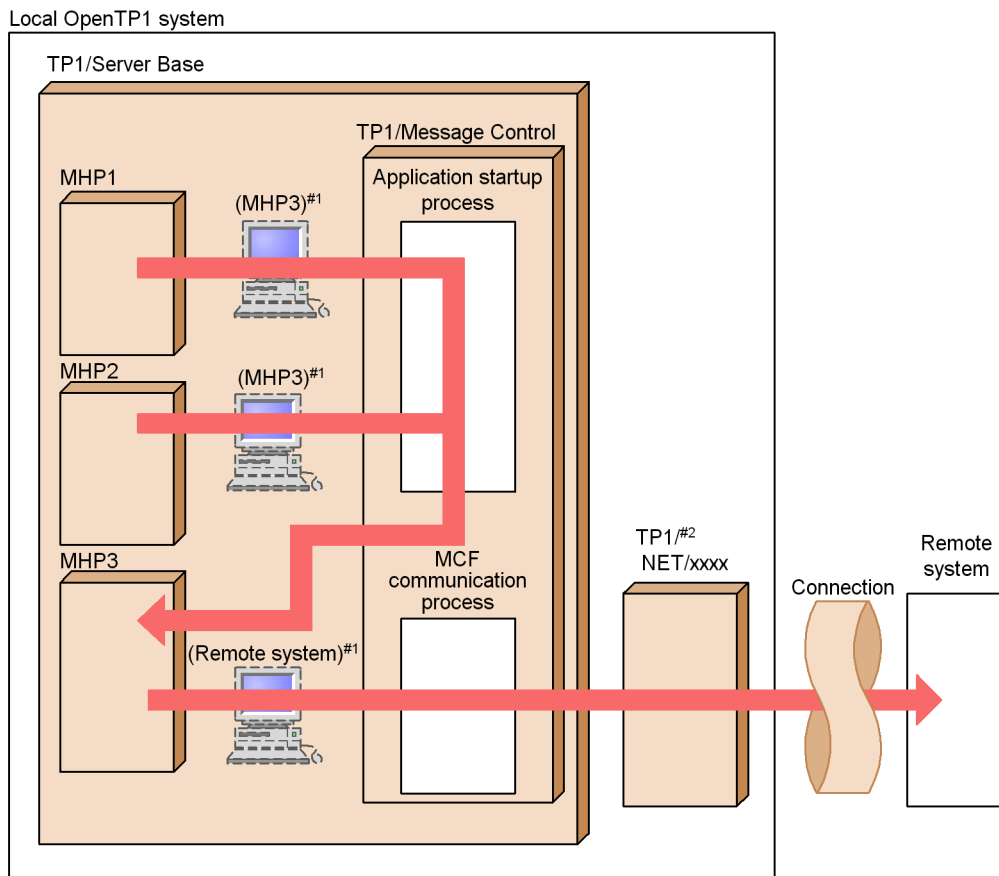
If you write an MHP that edits and sends a response message, you can use that MHP to respond to the requesting logical terminal instead of using the MHP that received the inquiry message. If you write an MHP that edits and sends a send-only message, you can use that MHP to send a message by starting the MHP from another MHP or SPP.

In both the above cases, message sending can be unified by issuing the `dc_mcf_execap()` function from the UAP that requested message sending.

The `dc_mcf_execap()` function is an asynchronous communication function. MCF starts an MHP that edits and sends messages immediately after the normal termination of the service of the MHP or SPP that issued the `dc_mcf_execap()` function, or when a specified time has expired (timer start). Therefore, the MHP or SPP that requested the message sending and the MHP that edits and sends messages are executed in different global transactions. The MHP or SPP that requested message sending uses MCF to pass necessary data (such as the name of the target logical terminal) to the MHP that edits and sends messages. In the case of a timer start, you can use the `mcfadltap` command or the `dc_mcf_adltap` function to cancel the start request.

Figure 3-37 explains how to unify the sending of messages. In the diagram, note that TP1/NET/xxx indicates the product required for the protocol being used.

Figure 3-37: Message sending unified by the dc\_mcf\_execap() function



#1: Parentheses enclose the logical terminal name.  
 #2: Product supporting the communication protocol.

#### (4) Handling MCF messages when OpenTP1 terminates

##### (a) Monitoring of unprocessed messages

MCF monitors the output queue from the start of termination processing till the last message in the output queue is processed. This monitoring prevents the termination processing from taking too much time because of unprocessed messages left in the output queue. The period for monitoring the output queue during OpenTP1 processing is called the *lifetime for unprocessed send messages*. If messages remain in the output queue after the lifetime expires, OpenTP1 discards them and the MCF event that reports discarding of an unprocessed send message (ERREVTA) occurs. If there is an MHP for this MCF event, MCF starts that MHP. Response messages, however, are not

included in the lifetime for unprocessed send messages. If response messages remain when the system terminates, such response messages are unconditionally deleted and ERREVT<sub>A</sub> does not occur.

MCF also monitors the period from the time OpenTP1 starts termination processing to the time that all the messages in the input queue are processed. This monitoring prevents the termination processing from not ending in situations where an error in the MHP results in the messages in the input queue not being processed. When the system terminates normally, OpenTP1 discards messages that remain due to the shutdown of a service group, and then ERREVT<sub>2</sub> occurs. The period for monitoring the input queue during OpenTP1 termination processing is called the *lifetime for unprocessed received messages*. If messages remain in the input queue after the lifetime expires, OpenTP1 assumes that the MHP has some abnormality and MCF terminates in an abnormal termination mode.

In the MCF communication configuration definition, you can specify the lifetimes for unprocessed send messages and unprocessed received messages.

### (b) OpenTP1 termination modes and unprocessed messages

OpenTP1 has five termination modes:

- normal termination
- forced normal termination
- planned termination A
- planned termination B
- forced termination

When OpenTP1 terminates in the *normal termination mode* or in the *forced normal termination mode*, OpenTP1 refuses to accept new messages and continues processing until all the messages in the input and output queues are processed. OpenTP1 also monitors the lifetimes of unprocessed received messages and unprocessed send messages.

When OpenTP1 terminates in the *planned termination A mode*, OpenTP1 refuses to accept new messages and continues processing until all the messages in the input queue have gone. OpenTP1 terminates even if messages remain in the output queue, but the unprocessed messages are processed the next time OpenTP1 starts. Note that when `reruntm=no` is specified in the `-o` option of the `mcf tpsvr` command in the MCF communication configuration definition, messages in a memory output queue and messages for timer start requests cannot be restored. In this mode, OpenTP1 monitors the lifetime for unprocessed received messages only.

When OpenTP1 terminates in the *planned termination B mode*, OpenTP1 completes the currently executing service, then terminates. Since the messages in the input and output queues are left unprocessed, the lifetime of unprocessed messages is not

monitored. The messages left in the input and output queues are processed the next time OpenTP1 starts. Note that when `reruntm=no` is specified in the `-o` option of the `mcftpsvr` command in the MCF communication configuration definition, messages in a memory queue and messages for timer start requests cannot be restored.

When OpenTP1 terminates in the *forced termination mode*, OpenTP1 terminates without waiting for the end of the currently executing service. The lifetime of the unprocessed messages is not monitored. The messages left in the input and output queues are processed the next time OpenTP1 starts. Note that when `reruntm=no` is specified in the `-o` option of the `mcftpsvr` command in the MCF communication configuration definition, messages in a memory queue and messages for timer start requests cannot be restored.

Table 3-6 summarizes how each termination mode processes the messages in the input and output queues.

*Table 3-6:* Termination modes and processing of messages in input and output queues

Termination mode	Messages in the input queue	Messages in the output queue
Normal termination	OpenTP1 processes all the messages.	OpenTP1 processes all the messages.
Forced normal termination	OpenTP1 processes all the messages.	OpenTP1 processes all the messages.
Planned termination A	OpenTP1 processes all the messages.	OpenTP1 leaves messages in the queue.
Planned termination B	OpenTP1 leaves messages in the queue.	OpenTP1 leaves messages in the queue.

### 3.3.10 Order of sending MCF messages

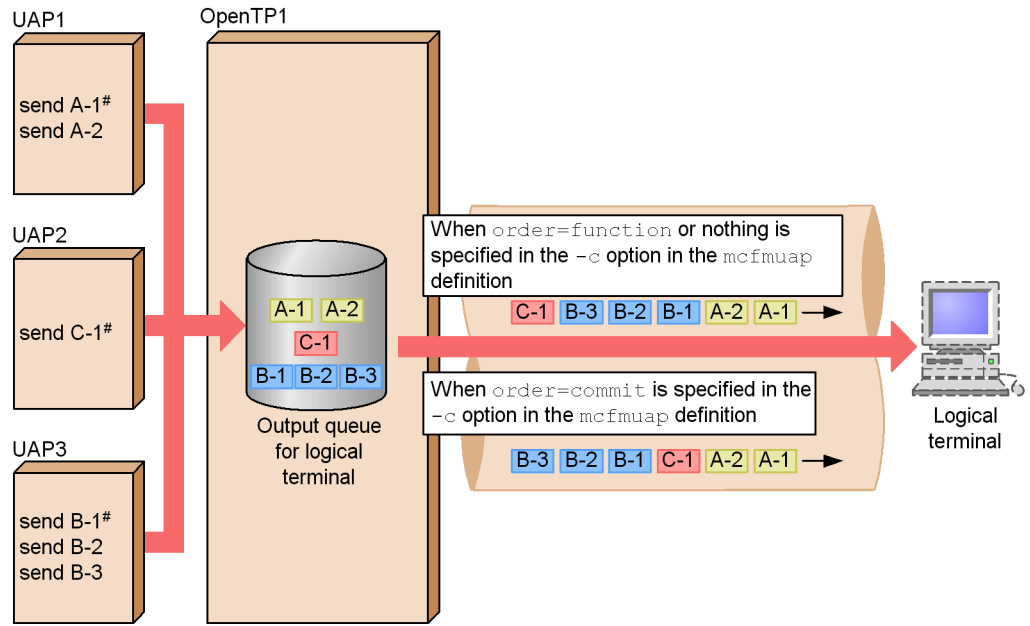
OpenTP1 schedules message transmission to a remote system as described below.

#### (1) Order of sending MCF messages to logical terminals

The order in which OpenTP1 sends messages to connected logical terminals is based on a combination of *first-in first-out (FIFO)* and message priority.

The segments in a message are sent continuously. After one message has been sent, the next message is sent to a logical terminal. You use the `order` operand of the `-c` option in the UAP common definition (`mcfmuaap`) to specify the order in which transmission requests from multiple UAPs are to be processed. Figure 3-38 illustrates the sending order.

Figure 3-38: Order of sending MCF messages to logical terminals



Legend:

N-n Message (segment)

#: The send requests occur in the order of A, B, and C.

The order in which UAPs are committed is UAP1, UAP2, and UAP3.

## (2) Relationship between message type and sending priority

The type of message to be sent by the UAP determines its priority. When different types of messages are waiting to be sent to the same logical terminal, the message with the highest priority is sent first. When the same type of messages are waiting, they are sent in FIFO order.

Table 3-7 lists the *sending priority* of the different types of messages.

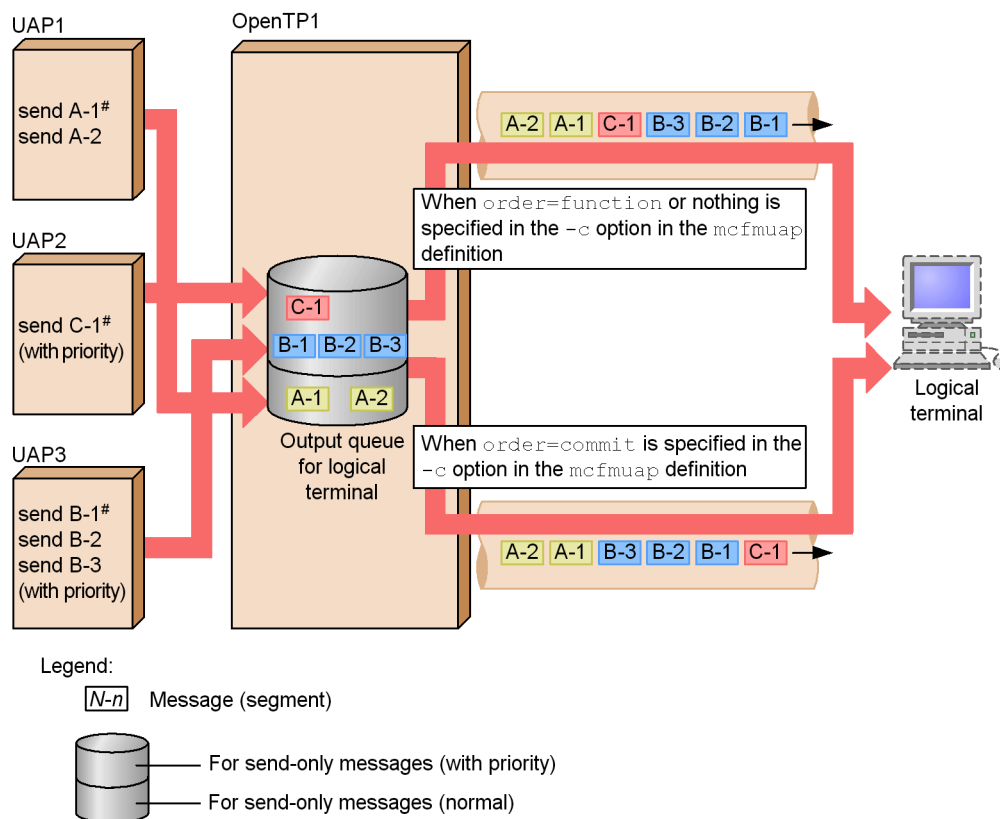
Table 3-7: Message type and sending priority

Message type	Sending priority
Response message	High
Send-only message (with priority)	Medium
Send-only message (normal) or inquiry message	Low

Figure 3-39 illustrates the order in which OpenTP1 sends messages to another system.

In the diagram, messages are written to the output queues in the order of A-B-C; however, the messages are sent to the logical terminals in the order of B-C-A. B and C are sent before A because they have higher sending priorities. B is sent before C based on the FIFO method: that is, B was written to the output queue before C.

Figure 3-39: Sending order based on sending priority combined with FIFO



#: The send requests occur in the order of A, B, and C.

The order in which UAPs are committed is UAP1, UAP2, and UAP3.

### 3.3.11 Partially changing the MCF communication service during operation of OpenTP1

You can change the `main()` function, user exit routines, and libraries for the MCF communication service without stopping OpenTP1 when you use any of the following products that are for supporting a communication protocol:

- TP1/NET/TCP/IP
- TP1/NET/Secondary Logical Unit - TypeP2

- TP1/NET/OSAS-NIF

You can change the `main()` function, user exit routines, and libraries, without stopping OpenTP1, by stopping only the desired part of the MCF communication service to replace the desired files. Since this method allows the other parts of the MCF communication service to run without interruption, the OpenTP1 system can continue to operate. You can replace the following files:

- The load module (`main()` function) of the MCF communication service
- User-created libraries (user exit routines) used by the MCF communication service
- The library provided by the used communication protocol product

To partially change the MCF communication service, you must use the operation commands (the `mcf_tlscom` command, the `mcf_tstart` command, and the `mcf_tstop` command) for managing the MCF communication service. For details about these commands, see the manual *OpenTP1 Operation*.

When the OpenTP1 system uses an MCF independent restart or remote MCF service, you cannot partially change the MCF communication service.

### 3.3.12 MCF capabilities that are not supported in Windows

In Windows, the MCF capabilities listed in the following table are not supported.

Table 3-8: MCF capabilities that are not supported in Windows

Capability	Description
User exit routine that determines inheritance of timer-start messages	--
MCF on-line tester	--
Specification of whether to acquire OJ statistics for an SPP	The <code>mcf_spp_oj</code> operand in the user service definition cannot be specified.
Statistics acquisition	<code>mcfmcomn -w stat</code> in the MCF manager definition cannot be specified. The <code>mcfreport</code> command cannot be used.
Speeding up MCF start processing when a memory queue is used as the input queue	<code>mcftevn -q diskitq</code> in the MCF communication configuration definition cannot be specified.

Legend:

--: Not applicable



## 3.4 Scheduling

The scheduling facility provided by OpenTP1 controls schedules and processes to efficiently process service requests to an SPP or MHP.

### 3.4.1 Scheduling requests to service-providing programs

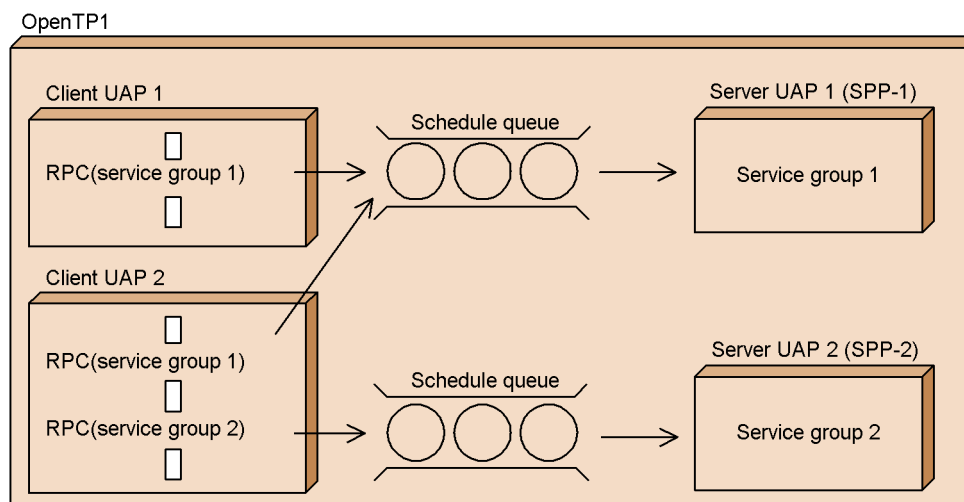
#### (1) Schedule queues for SPPs

OpenTP1 uses *schedule queues*, which are created for each service group of an SPP, to schedule service requests to an SPP. When a UAP requests a service from an SPP, OpenTP1 catalogs the service request in the proper schedule queue by using the service group name and service name specified in the client UAP's RPC. Scheduling of service requests is possible even if a service group of the SPP specified by the client UAP is located on another node in the network.

Service requests can be assigned scheduling priorities. Service requests with higher scheduling priorities are taken out from the schedule queue first; service requests with the same scheduling priorities are taken out with the FIFO method.

Figure 3-40 illustrates scheduling of service requests for SPPs.

Figure 3-40: Scheduling of service requests for SPP



#### (2) Setting scheduling priorities for service requests

The programmer can assign a *scheduling priority* to a service request issued from a client UAP. The scheduling priority for a service request is declared by writing the `dc_rpc_set_service_prio()` function immediately before the `dc_rpc_call()`

function for the RPC that a client UAP issues. Based on the scheduling priority declared here, the schedule service at the server UAP schedules the service requests.

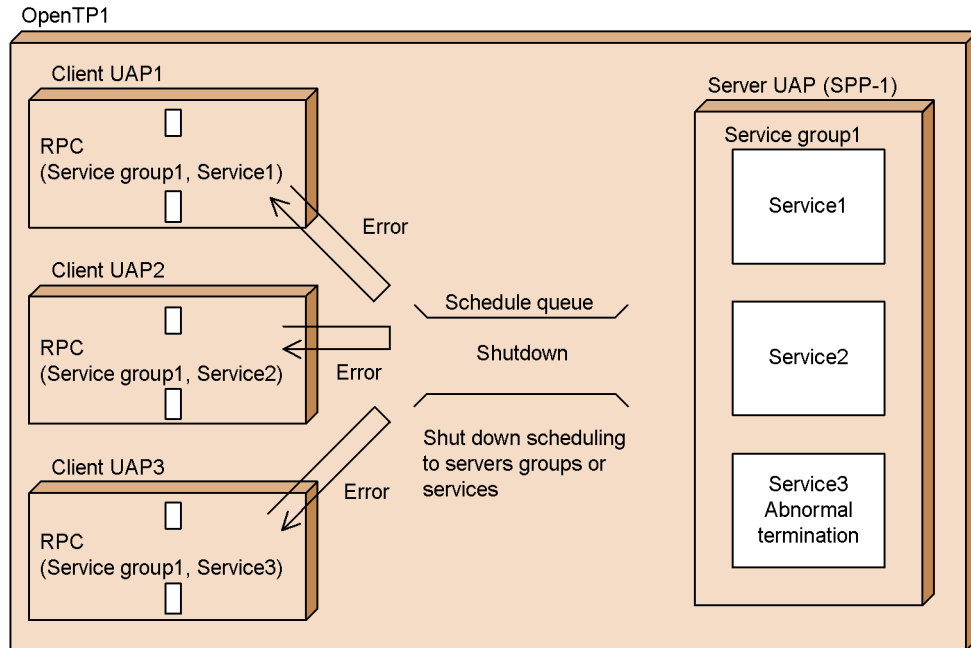
In the user service definition for the server UAP (the SPP), you can specify whether or not scheduling is to follow the scheduling priorities of the client UAPs.

### (3) Shutting down scheduling for an SPP

In the user service definition, you can specify whether scheduling of service requests for an SPP should be blocked when the SPP abnormally terminates while executing a service. In OpenTP1 this is called *shutting down scheduling of service requests to an SPP*.

Figure 3-41 illustrates shutting down scheduling of service requests to a service group.

Figure 3-41: Shutdown of scheduling of service requests to a service group



#### (a) Specifiable target when shutting down scheduling

You can specify whether OpenTP1 is to shut down scheduling of service requests to service groups or to services. When an SPP service abnormally terminates:

- if you have specified shutdown of scheduling for service groups, OpenTP1 immediately shuts down scheduling of all requests to the SPP.
- if you have specified shutdown of scheduling for services, OpenTP1 shuts down scheduling of requests to the affected service only but does not shut down

scheduling of requests to other services in the same service group.

**(b) Processing when an SPP terminates abnormally**

If you have specified shutdown of scheduling for service groups and a UAP process in an SPP abnormally terminates, OpenTP1 shuts down scheduling of requests to all services in that service group. After scheduling of requests to a service group is shut down and a request for a service arrives, an error is returned to the client UAP that issued the request. If a service request is already in the schedule queue when scheduling of service requests to a service group is shut down, an error is also returned to the client UAP that issued the request.

When no scheduling shutdown is specified for when a UAP process in an SPP abnormally terminates, scheduling proceeds without shutdown. However, if there are three or more abnormal terminations within the period specified in the user service definition, then scheduling is shut down for the service group of the SPP.

**(c) Using a command to release SPP shutdown**

You can use the command `scdrls` to release the scheduling shutdown for an SPP and restart the SPP.

**(d) How scheduling-shutdown status is handled at restart**

In the user service definition for each SPP, you can specify whether an SPP is to inherit the SPP scheduling-shutdown status when a complete recovery is performed following a stoppage of the online system. Among those SPPs for which scheduling was shut down when online processing stopped, the SPPs for which inheritance of scheduling-shutdown status was specified are restarted with the same scheduling-shutdown status even after a complete recovery. An error is returned to the source of a request for a service of an SPP for which scheduling has been shut down.

For those SPPs for which no inheritance of scheduling-shutdown status is specified, the SPP is restarted with the status at the release of the shutdown, and the requested service is executed.

**(e) Using commands to shut down scheduling**

You can use the command `scdhold` to shut down scheduling and use the command `scdrls` to release scheduling shutdown, thereby enabling SPPs to be modified online.

The command `scdhold` can shut down scheduling of service requests to service groups or to services. To replace an SPP online, the OpenTP1 administrator should shut down scheduling to service groups of the SPP, replace the SPP with a new SPP, and then use the `scdrls` command to release the scheduling shutdown and execute the new SPP.

If you specify the `-p` option with the server name in the `scdhold` command (`scdhold -s server-name -p`), even after scheduling for an SPP has been shut down, service

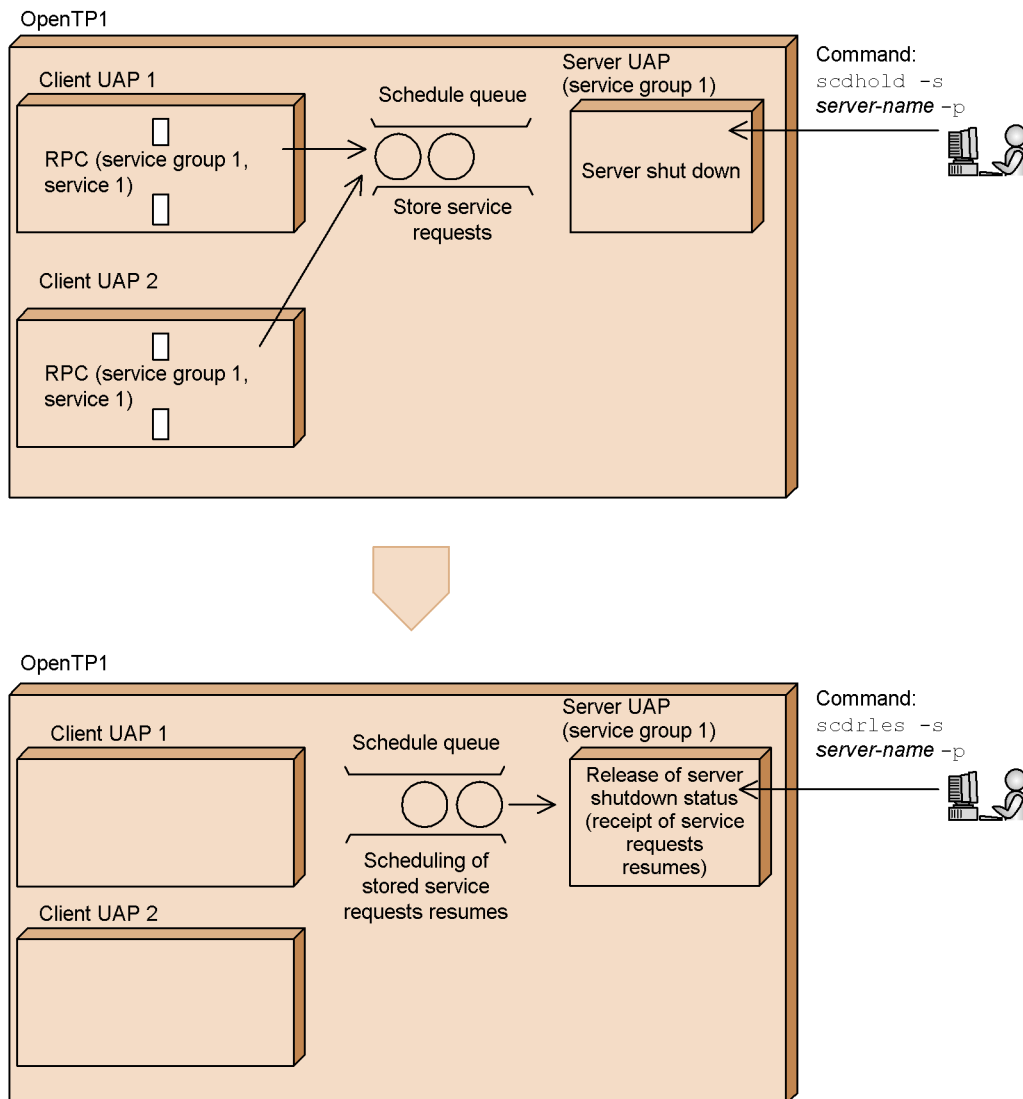
### 3. Functions

requests for the SPP can still be received. The received requests are stored in the schedule queue. When the SPP is again in a condition to process services, OpenTP1 restarts scheduling for the stored service requests.

Storage of service requests is only possible for shutting down scheduling of service requests to service groups. When scheduling of service requests to services is shut down, you cannot use the `scdhold` command to store service requests.

Figure 3-42 illustrates service requests stored in the schedule queue when scheduling is shut down.

Figure 3-42: Shutting down scheduling and storing service requests in the schedule queue



#### (4) Server that receives requests from a socket

In order to improve scheduling efficiency, you can specify an SPP that is to receive services without going through the schedule queue. This kind of UAP (a user server) is called a *server that receives requests from a socket*. Service requests are passed directly to a server (SPP) that receives requests from a socket; the requests are not passed through a schedule queue.

In cases where the server that receives requests from a socket cannot receive service requests, such as when waiting for a transaction synchronous point, OpenTP1 temporarily stores the service requests. When the server that receives requests from a socket can once again receive service requests, OpenTP1 passes the service requests to the server. When the area used for temporary storage is full, errors are returned to the source of the service requests.

In the user service definition, you can specify whether an SPP is to receive requests from a socket.

Compared to a *server (SPP) that receives requests from the queue*, the following restrictions apply to a server that receives requests from a socket:

- Operation commands cannot be used to shutdown scheduling.
- The Multiserver facility (described in *3.4.3 Process control and the Multiserver facility*) cannot be used. In the user service definition, you can specify one resident process only for the number of usable processes.

You can activate more than one server that receive requests from the socket only by changing the user server name (file name of the user service definition). In this case, specify the same service group name, server name, and executable program name.

When more than one server that receive requests from the socket are activated, service requests can be distributed as with a Multiserver facility.

Note that a server that receives requests from the socket and a server that receives requests from the queue cannot be activated concurrently if their service group names are the same. If a server that receives requests from the queue of the same service group has already been activated, terminate it before activating the server that receives requests from the socket.

For details of the Multiserver facility, see *3.4.3(1) Multiserver facility*.

### **(5) Scheduling control of services**

By specifying the `scdsvcdef` definition command in the user service definition or user service default definition, you can specify the following scheduling controls for a specific service requested via the schedule queue for an SPP. For details about the `scdsvcdef` definition command, see the manual *OpenTP1 System Definition*.

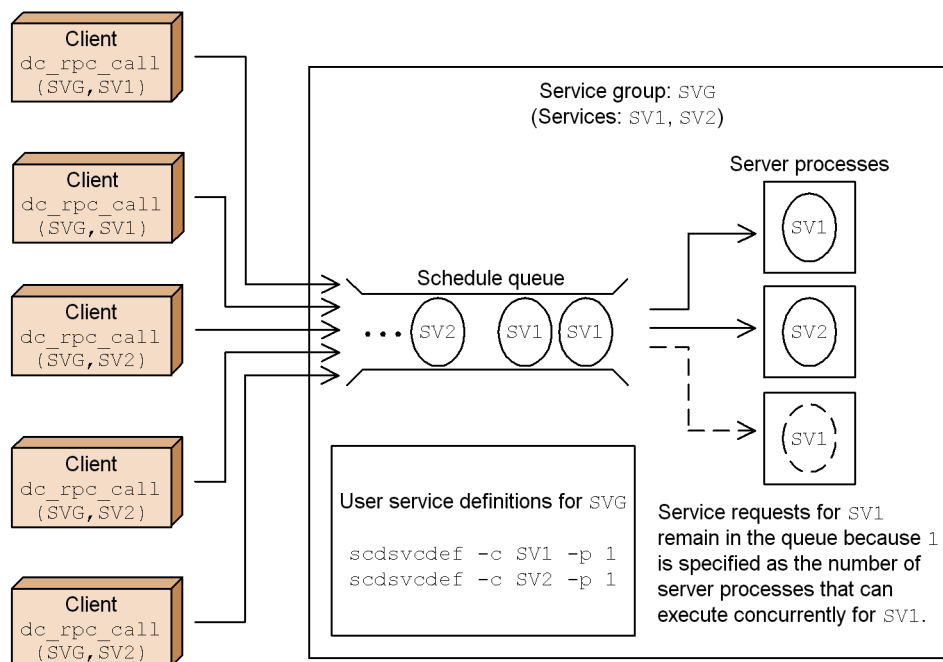
- Maximum number of server processes that can be executed concurrently for a specific service
- Maximum number of requests for a specific service that can be placed in the schedule queue
- Maximum size of the message-storing buffer pool for a specific service that can be placed in the schedule queue

**(a) Maximum number of server processes that can be executed concurrently for a specific service**

If the number of server processes that are processing a particular service reaches the maximum number of concurrent executions specified in the `scdsvcdef` definition command, requests for those services whose set number of concurrently executable processes has not yet been reached are kept scheduled, regardless of their order of placement in the queue and their scheduling priority.

If the queued service requests are all for services whose maximum number of concurrently executable processes has already been reached, the pending requests wait for completion of the service processing currently being executed. Figure 3-43 illustrates the maximum number of server processes that can be executed concurrently.

*Figure 3-43: Maximum number of server processes that can be executed concurrently for a specific service*



Legend:

- `...` : Indicates that further service requests are waiting to be placed in the schedule queue.
- `→` : Indicates that service requests were queued successfully or were removed from the queue.
- `->` : Indicates that service requests could not be queued or were removed from the queue.

1. Specify the `-c` and `-p` options of the `scdsvcdef` definition command as follows. This sets the maximum number of server processes that can be executed

concurrently for services SV1 and SV2 as 1 in each case.

```
scdsvcddef -c SV1 -p 1
scdsvcddef -c SV2 -p 1
```

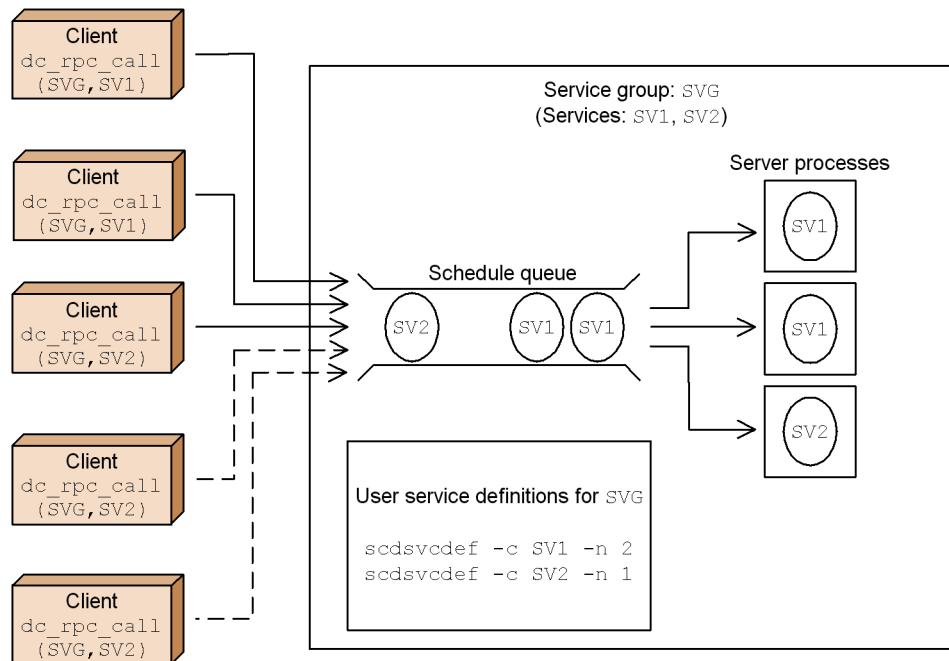
- Service requests in excess of the limit placed on the number of concurrently executable server processes are left in the schedule queue.

**(b) Maximum number of requests for a specific service that can be placed in the schedule queue**

If the number of service requests for a particular service reaches the maximum number of queued requests specified in the `scdsvcddef` definition command, subsequent requests for that service are re-directed to another OpenTP1 node instead of the local schedule queue.

If the subsequent requests cannot be rescheduled on the other OpenTP1 node, an error is returned to the client UAP, indicating that the message-storing buffer is full. Figure 3-44 illustrates the maximum number of queued requests for a specific service.

*Figure 3-44: Maximum number of requests for a specific service that can be placed in the schedule queue*



Legend:

- : Indicates that service requests were queued successfully or were removed from the queue.
- > : Indicates that service requests could not be queued or were removed from the queue.



1. Specify the `-c` and `-n` options of the `scdsvcdef` definition command as follows. This sets the maximum number of requests that can be queued as 2 for SV1 and as 1 for SV2.

```
scdsvcdef -c SV1 -n 2  
scdsvcdef -c SV2 -n 1
```

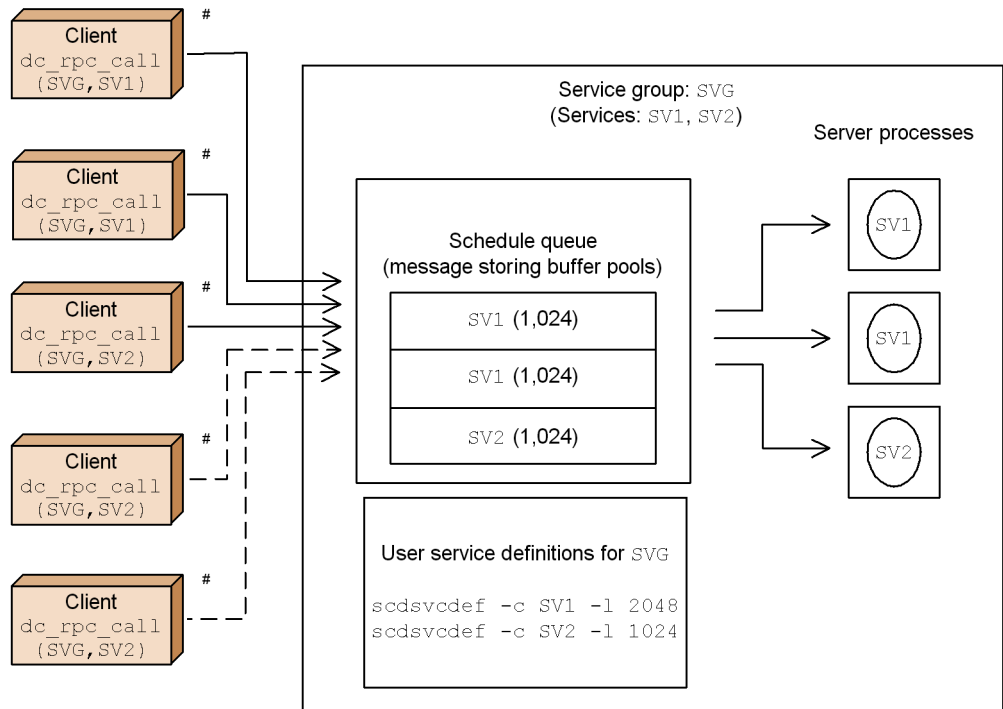
2. Service requests in excess of the limit placed on the number of queued requests cannot be scheduled.

**(c) Maximum size of the message-storing buffer pool for a specific service that can be placed in the schedule queue**

If the length of the message-storing buffer pool containing service requests for a particular service reaches the maximum length specified in the `scdsvcdef` definition command, subsequent requests for that service are re-directed to another OpenTP1 node instead of the local schedule queue.

If the subsequent requests cannot be rescheduled on the other OpenTP1 node, an error is returned to the client UAP, indicating that the message-storing buffer is full. Figure 3-45 illustrates the maximum size of the queued message-storing buffer pool for a specific service.

*Figure 3-45: Maximum size of the message-storing buffer pool for a specific service that can be placed in the schedule queue*



Legend:

- > : Indicates that service requests were queued successfully or were removed from the queue.
- > : Indicates that service requests could not be queued or were removed from the queue.

#: RPC message size is assumed to be 1,024 bytes.

1. Specify the `-c` and `-l` options of the `scdsvcdef` definition command as follows. This sets the maximum size of the queued message-storing buffer pool as 2048 for SV1 and as 1024 for SV2.
 

```
scdsvcdef -c SV1 -l 2048
scdsvcdef -c SV2 -l 1024
```
2. Service requests in excess of the limit placed on the size of the queued message-storing buffer pool cannot be scheduled.

### 3.4.2 Scheduling MCF messages to message-handling programs

#### (1) Schedule queues for MHPs

OAs with SPPs, OpenTP1 schedules messages to MHPs by creating a *schedule queue*

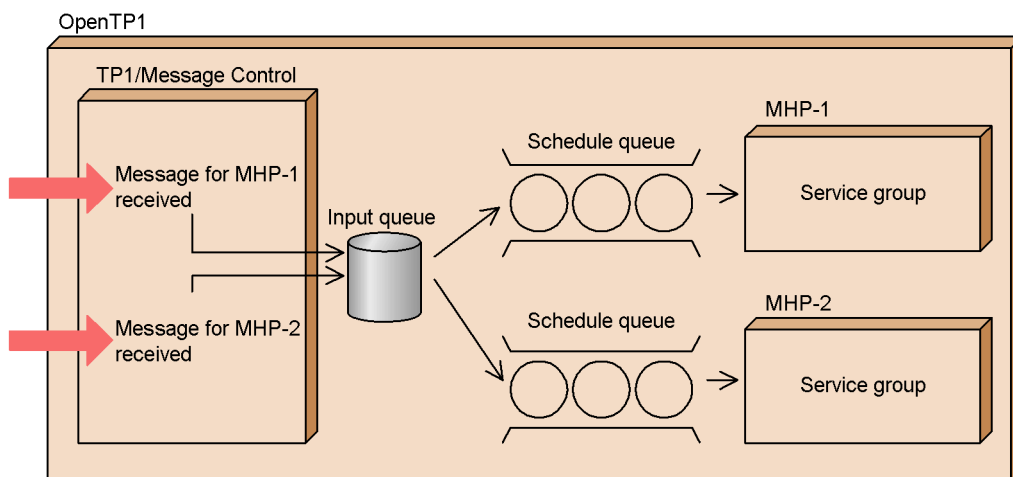
for each MHP service group.

First MCF converts the MCF-application name included in the message into the service group name and the service names, and then catalogs the received message into the input queue. When the last segment of a received message is stored in the input queue, MCF catalogs the messages into the proper schedule queue depending on the correspondence between the MCF application and the service group name and service names defined in the application attribute definition. Scheduling fails unless the MHP service group and the MCF that received the message are located on the same node.

Messages to an MHP are taken out from the schedule queue by the FIFO method.

Figure 3-46 shows MHP scheduling.

*Figure 3-46: Scheduling messages for an MHP*



## (2) Shutting down scheduling using commands

### (a) Shutting down scheduling using OpenTP1 commands

An OpenTP1 administrator can use commands to shut down scheduling of service requests to an MHP, and to release such scheduling shutdowns. As shown in the following table, these commands can apply to the scheduling of service requests to MCF applications, services, or service groups.

#### ■ Commands for shutting down scheduling

`mcfadctap`: Shut down scheduling of service requests to a specific MCF application

`mcftdctsg`: Shut down scheduling of service requests to a specific service group

`mcftdctsv`: Shut down scheduling of service requests to a specific service

- **Commands for releasing scheduling-shutdown status**

`mcfactap`: Release the scheduling-shutdown status of a specific MCF application

`mftactsg`: Release the scheduling-shutdown status of a specific service group

`mftactsv`: Release the scheduling-shutdown status of a specific service

The user can specify whether the service group will become active or remain shut down after complete recovery from an abnormal termination. The user can make this decision either when the user uses the `mftdctsg` command to shut down the service group, or when using the status inheritance definition to create the system.

- **Using `mcfadctap` to shut down scheduling for MCF application**

The scheduling-shutdown status of an MHP shut down with `mcfadctap` can be reproduced at a full recovery after a stoppage of the online system. In the status inheritance definition, you can specify whether the scheduling-shutdown status is to be inherited.

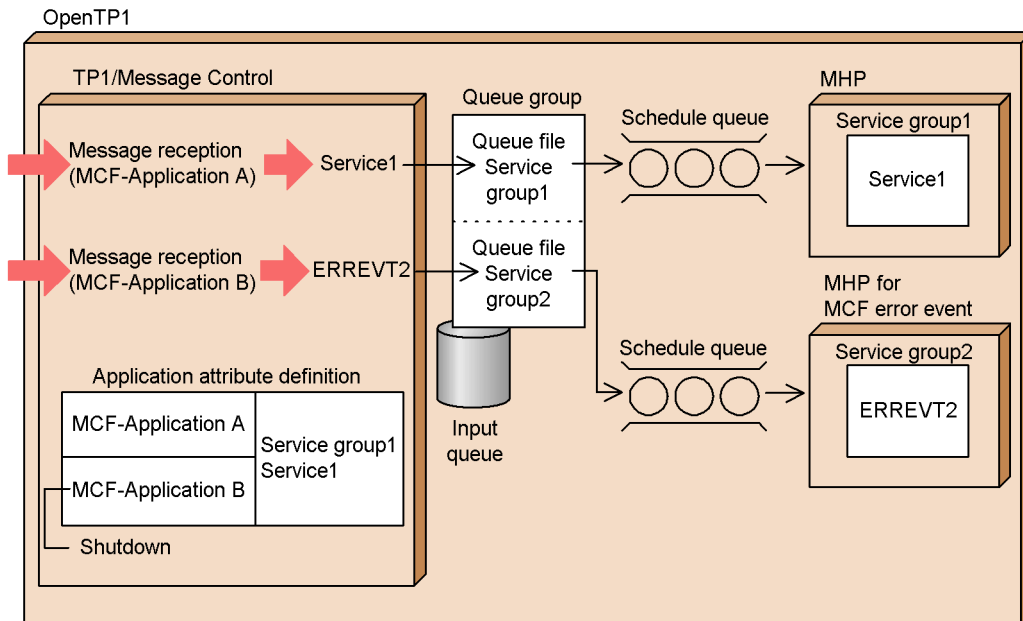
Table 3-9 shows the two types of scheduling shutdown possible by specifying an MCF-application name.

*Table 3-9: Shutting down scheduling of service requests to MCF-applications*

Type of scheduling shutdown	Processing of messages that arrived after the shutdown	Processing of messages already cataloged in the input queue before the shutdown
Preventing both entry to input queue and output from schedule queue	The message is cataloged in the input queue for the MHP for ERREVT2 (an MCF event).	The message is recataloged in the input queue for the MHP for ERREVT2 (an MCF event).
Preventing entry to input queue only	The message is cataloged in the input queue for the MHP for ERREVT2 (an MCF event).	The relevant MHP is started normally.
Preventing only scheduling	The message is cataloged in the input queue for the MHP for ERREVT2 (an MCF event).	The message is recataloged in the input queue for the MHP for ERREVT2 (an MCF event).

For each communication protocol, you can assign a different MCF-application name for a service name. By the use of different MCF-application names in different protocols, you can ensure that when an MCF-application name is used for a shutdown, only the service requests from one specific protocol will be refused. Figure 3-47 illustrates scheduling shutdown by specifying the MCF-application name.

Figure 3-47: Shutting down scheduling by specifying an MCF application name



- Using mcfstdctsv to shut down scheduling for services

The two types of scheduling shutdown (i.e., preventing entry to the input queue or output from the schedule queue) are also possible when shutting down scheduling of service requests to services. Table 3-10 shows the methods for shutting down scheduling of service requests to MHP services.

Even if an MHP service is shut down, other services in the service group are still usable.

Table 3-10: Shutting down scheduling of service requests to MHP services

Type of scheduling shutdown	Processing for messages that arrived after the shutdown	Processing for messages already cataloged in the input queue before the shutdown
Preventing both entry to input queue and output from schedule queue	The message is cataloged in the input queue for the MHP for ERREVT2 (an MCF event).	The message is recataloged in the input queue for the MHP for ERREVT2 (an MCF event).
Preventing entry to input queue only	The message is cataloged in the input queue for the MHP for ERREVT2 (an MCF event).	The MHP is started normally.

Type of scheduling shutdown	Processing for messages that arrived after the shutdown	Processing for messages already cataloged in the input queue before the shutdown
Preventing only scheduling	The message is cataloged in the input queue for the MHP for ERREVT2 (an MCF event).	The message is recataloged in the input queue for the MHP for ERREVT2 (an MCF event).

- Using mcftdctsg to shut down scheduling for service groups

The shutting down of scheduling of service requests to a service group can be classified into two types depending on whether the input queue is a disk queue or a memory queue. Each of these two types of scheduling shutdown can be further classified into three subtypes. Table 3-11 shows the scheduling shutdown of service requests to MHP service groups.

Table 3-11: Shutdown of MHP service group

Type of scheduling shutdown		Processing for messages that arrived after the shutdown	Processing for messages already cataloged in the input queue before the shutdown
Input queue is disk queue	Preventing both entry to input queue and output from schedule queue	The message is cataloged in the input queue for the MHP for ERREVT2 (an MCF event).	The message is placed in the schedule wait state. After the scheduling shutdown is released, the MHP is started normally.
	Preventing entry to input queue only	The message is cataloged in the input queue for the MHP for ERREVT2 (an MCF event).	The MHP is started normally.
	Preventing output from schedule queue only	The message is cataloged in the input queue and the message is placed in the schedule wait state. <sup>#</sup>	The message is placed in the schedule wait state. After the scheduling shutdown is released, the MHP is started normally.
Input queue is memory queue	Preventing both entry to input queue and output from schedule queue	The message is cataloged in the input queue for the MHP for ERREVT2 (an MCF event).	The message is cataloged in the input queue for the MHP for ERREVT2 (an MCF event).
	Preventing entry to input queue only	The message is cataloged in the input queue for the MHP for ERREVT2 (an MCF event).	The MHP is started normally.

Type of scheduling shutdown		Processing for messages that arrived after the shutdown	Processing for messages already cataloged in the input queue before the shutdown
	Preventing output from schedule queue only	The message is cataloged in the input queue for the MHP for ERREVT2 (an MCF event).	The message is recataloged in the input queue for the MHP for ERREVT2 (an MCF event).

#

If the number of messages cataloged in the disk input queue exceeds the maximum, the messages that could not be stored are cataloged in the input queue for the MHP for ERREVT2 (an MCF event).

### (b) Shutting down scheduling automatically

In the application attribute definition, you can define whether or not scheduling for an MCF application or service should be shut down if an MHP terminates abnormally while executing a service. Also, in the application attribute definition, you can define that scheduling for an MCF application or service should be shut down after a specified number of abnormal terminations of the MHP. You can specify whether the number of abnormal terminations refers to the number of consecutive MHP abnormal terminations, or the number of MHP abnormal terminations in total.

To shut down in units of applications, use the `aplihold` command in the application attribute definition. To shut down in units of services, use the `servhold` command in the application attribute definition. To shut down in units of service groups, use the `srvghold` command in the application attribute definition.

Note that an abnormal termination includes the situation in which an MHP is rolled back by the rollback function `dc_mcf_rollback()` that has `no-return` specified in a parameter. For details of the rollback function, see the *OpenTPI Programming Guide*.

When shutting down only the scheduling in units of service groups, you can choose how to handle the received messages if MHP terminates abnormally while executing a service. You can:

- Replace received messages with an error event
- Reschedule received messages at the beginning of the schedule queue. To reschedule, assign `r` to `recvmsg` in the application attribute definition.

For automatic shutdown due to abnormal termination, the shutdown status of MHP is not inherited at a full recovery operation after a stoppage of the online system.

When the specification that prevents shutdown at abnormal termination is specified by

`aplihold`, `servhold`, or `srvghold` in the application attribute definition, the application, service, or service group will not be shut down. If frequent abnormal terminations occur with this specification, use a command to shut down the application, service, or service group.

**(c) Automatic scheduling shutdown caused by inconsistency between the defined MCF-application name and the MHP**

When MCF schedules an MHP, it checks for a service group that corresponds to the application name specified in the application attribute definitions. If no such service group exists, OpenTP1 shuts down scheduling for that service group only, and outputs a message to the message log to report that there is no corresponding service group. See Table 3-11 for differences between situations where the input queue is a disk queue and where the input queue is a memory queue.

The MHP scheduling-shutdown status, when scheduling for the MHP was automatically shut down because of an inconsistency between the MCF-application name and the MHP, is not inherited during a full recovery operation after a stoppage of the online system.

### 3.4.3 Process control and the Multiserver facility

OpenTP1 provides various facilities for controlling processes. Processes occur in memory that is used when OpenTP1 system services or user servers (UAPs) are executed. A process that is generated by executing a user server is sometimes called a *user server process*, a *UAP process* or simply a *process*.

In the process service definition, you can specify the total number of processes so that the number of system service processes and user server processes is neither more nor less than the number required.

You must start the user server before attempting to control user server processes. The user server can be started:

- automatically, at the same time as OpenTP1 starts; or
- manually, by the `dcsvstart -u` command.

The main facilities for controlling processes are:

- the Multiserver facility
- the Internode Load-Balancing facility

The following sections describe the above facilities in more detail.

#### **(1) Multiserver facility**

OpenTP1 provides the *Multiserver facility* to enable multiple instances of a service to be executed in parallel and in different processes, in response to multiple requests for the service. When a new service request comes to an already executing user server, a



new user server process can be executed for the request.

Not all SPPs can use the Multiserver facility. SPPs that use schedule queues (i.e., servers that receive requests from queues) and MHPs can use the Multiserver facility, but servers that receive requests from a socket cannot use the Multiserver facility. In the user service definition (in the `parallel_count` operand) specify that a server that receives requests from a socket should use one process only.

## **(2) Resident processes and non-resident processes**

A process of a UAP that uses the Multiserver facility can be reserved during OpenTP1 operations or can be reserved dynamically. Processes that are always reserved are called *resident processes*. Processes that are started when necessary are called *non-resident processes*.

An advantage of non-resident processes is that they enable efficient use of the memory area in the OpenTP1 system. An advantage of resident processes is that the processing for a user server is performed more quickly than for non-resident processes.

When using the Multiserver facility, in a user service definition you can specify the maximum number of processes to be used. If you specify more than one resident process, only that number of processes will be started and executed in parallel. If you specify more than one non-resident process, only that number of processes will be started dynamically.

If there is no spare system memory, a non-resident process is executed after the finish of any currently executing non-resident process.

Set the number of resident and non-resident processes in the `parallel_count` operand of the user service definition before starting the user server.

## **(3) Balancing loads in the Multiserver facility**

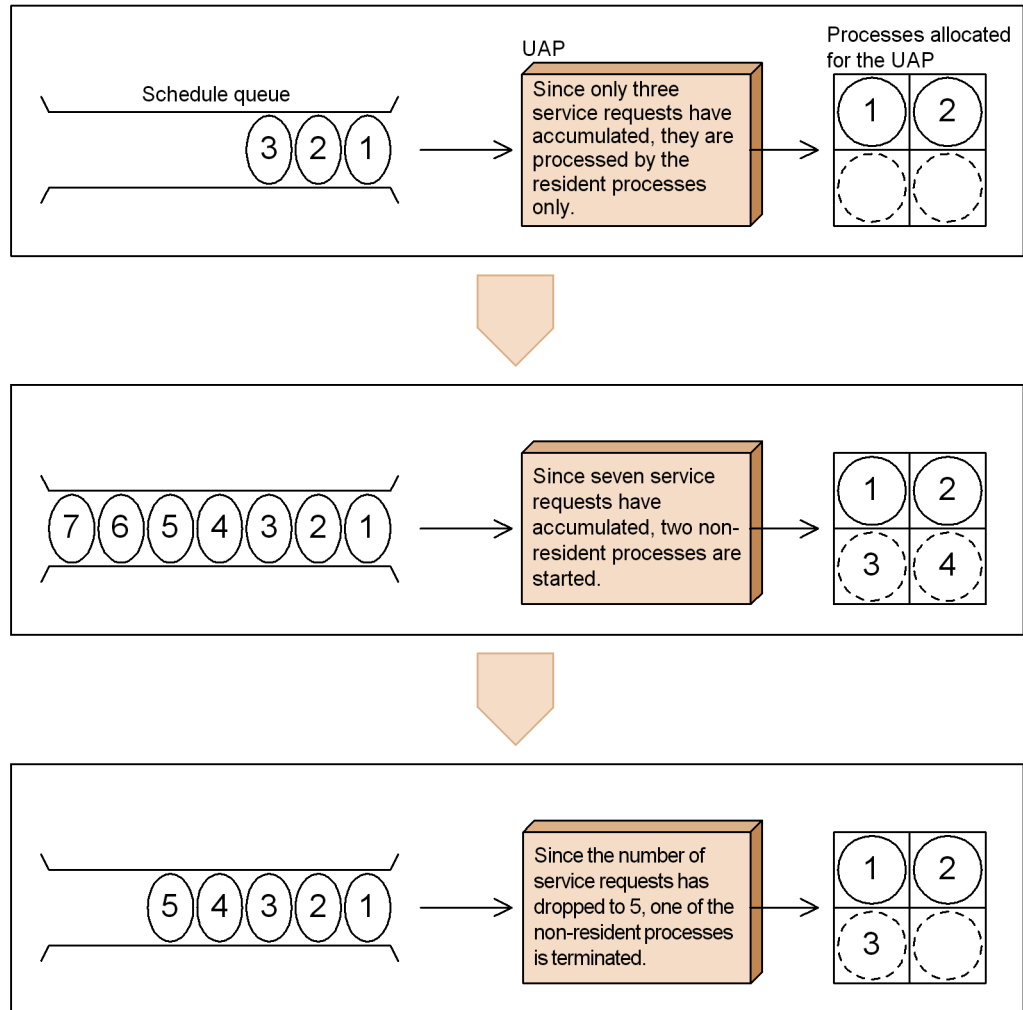
The Multiserver facility can increase or decrease the number of non-resident processes according to the number of service requests in the schedule queue.

The startup timing of non-resident processes depends upon the values specified in the `balance_count` operand in the user service definition. If there are remaining service requests that exceed (the values specified to the `balance_count` operand  $\times$  processes being started), OpenTP1 starts the non-resident processes. If the number of service requests remaining in a schedule queue is less than (the values specified to the `balance_count` operand  $\times$  processes being started), OpenTP1 will terminate the non-resident processes.

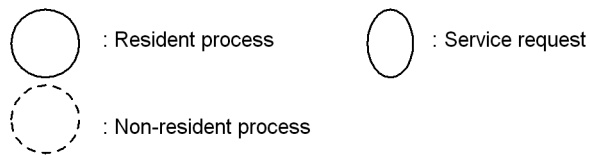
Figure 3-48 gives an overview of the Multiserver facility based on resident and non-resident processes.

*Figure 3-48: Overview of Multiserver facility based on resident and non-resident processes*

- Example with two resident processes, two non-resident processes, and a `balance_count` value of 2.



Legend:

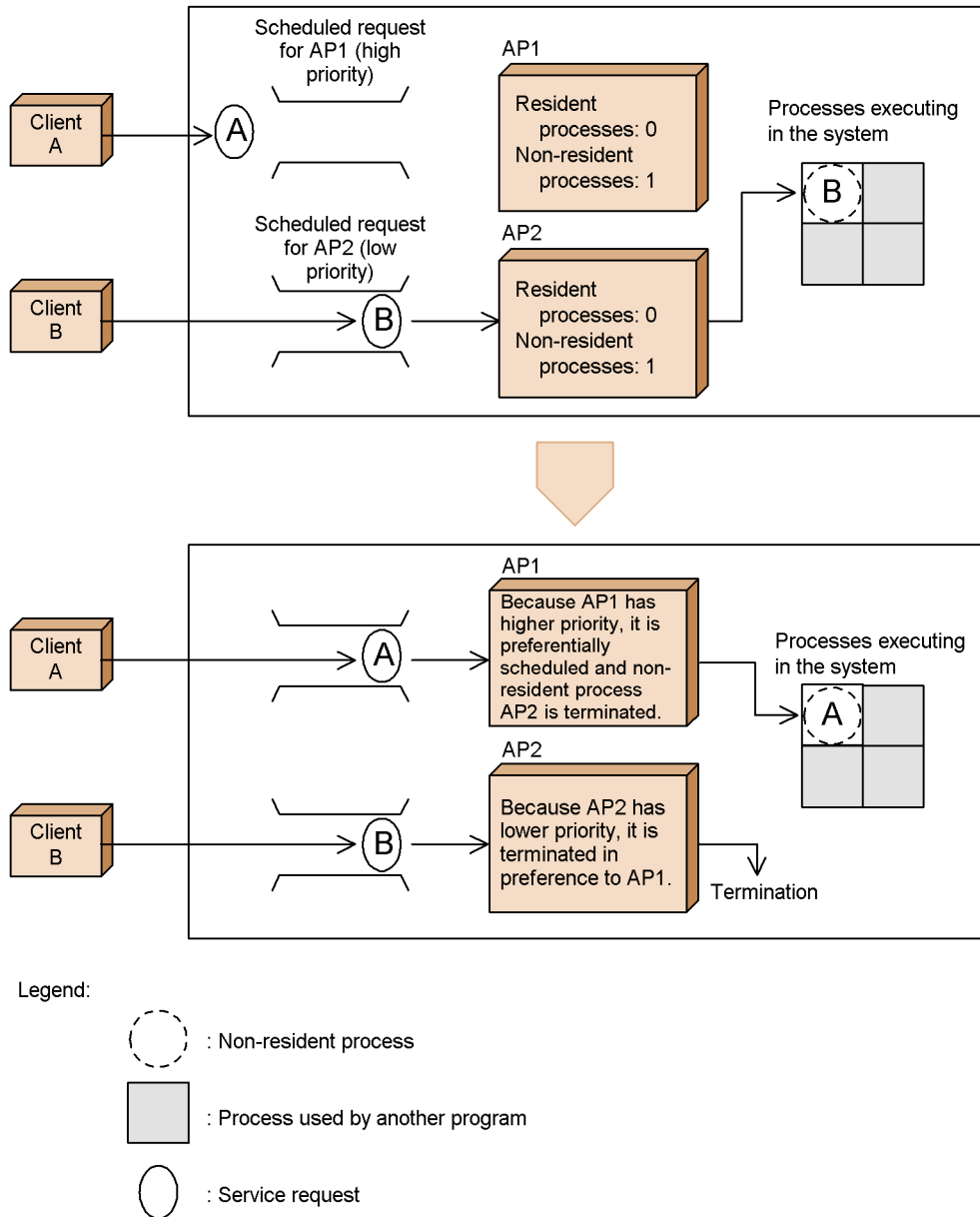


**(4) Scheduling priority for user servers**

In the user service definition, you can assign a scheduling priority to each user server. Non-resident processes in a user server that has high scheduling priority are preferentially scheduled compared to non-resident processes in a user server that has low scheduling priority.

Figure 3-49 provides an overview of scheduling priority.

Figure 3-49: Overview of scheduling priority



**(5) Balancing loads among nodes**

OpenTP1 can process a heavily-used service on multiple nodes. When many processes are required for processing a service requested from an SPP, OpenTP1 can distribute

the processing to SPPs with the same service group name on other nodes. To use this facility for balancing loads among nodes, the following conditions must be satisfied:

- A user server providing the same service to multiple nodes in the same LAN must be operating.
- In the `all_node` operand in the system common definition, each OpenTP1 node must define the other nodes. This definition must allow the OpenTP1 nodes to exchange the information (name information) of the user servers operating on the nodes.

Service requests are passed to the user server on a randomly selected node. OpenTP1 references the server information of the node and avoids selecting it if it is hard to schedule. Therefore, even when a schedulable user server exists on the local node, the request is not always passed to that user server. To select the user server on the local node first, specify `scd_this_node_first=Y` in the schedule service definition.

With `Y` specified, OpenTP1 selects a user server on another node only when it is hard for the user server on the local node to accept the request.

The internode load-balancing facility requires that the operation conditions of the user servers on nodes are the same. If the following conditions differ greatly from node to node that is selected, such an environment is unsuitable for the internode load-balancing facility. In this case, do not place the service groups with the same name in multiple nodes. The conditions that should not differ greatly are:

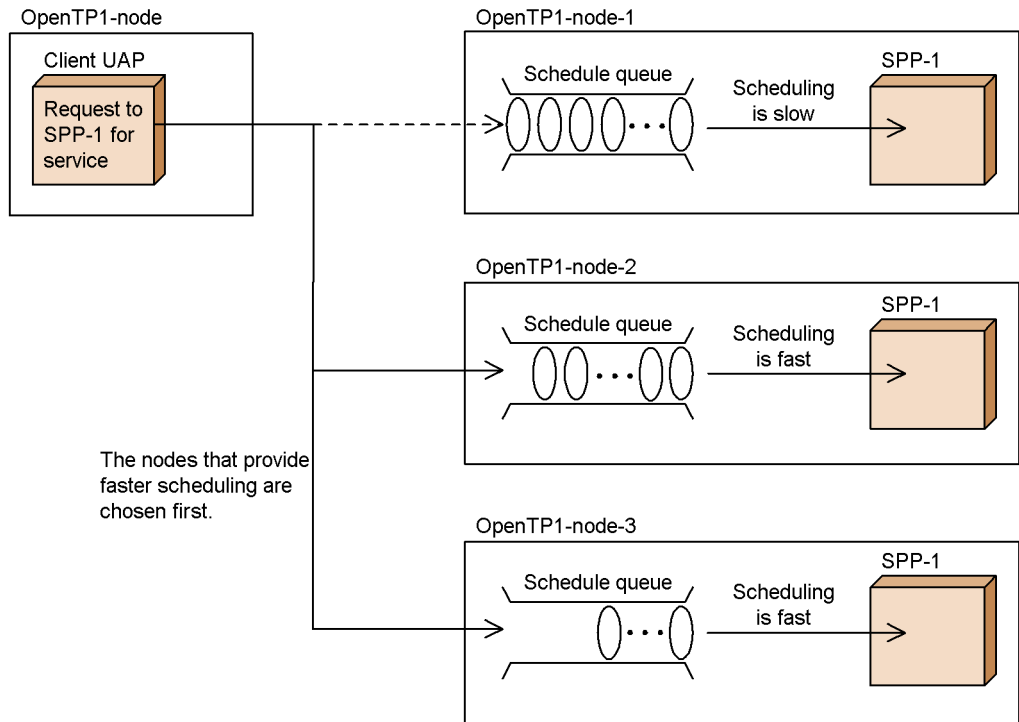
- Operation cost such as the connection fees of the public line
- Line speed
- Line quality
- Performance of individual nodes

Each node reports server information to be referenced when allocating a request to the user server. For a system in which SPPs of the same service group are not distributed among multiple nodes, the server information need not be reported. Especially when a public line is used, charges for unnecessary connections will occur. In such a system, specify `scd_announce_server_status=N` in the schedule service definitions of all the nodes to suppress reporting of server information.

OpenTP1 can distribute loads for SPPs (servers) that receive requests either from a queue or from a socket. When such an SPP is busy, OpenTP1 passes the service requests for the SPP to another user server in another node. The selection of the other node is almost random, except that for servers that receive requests from a queue, OpenTP1 checks the status of the node to be scheduled and controls selection so that it is difficult to select a node that has a low scheduling efficiency. For a server that receives requests from a socket, however, OpenTP1 neither checks the node status nor controls selection.

Figure 3-50 gives an overview of the Internode Load-Balancing facility.

Figure 3-50: Overview of Internode Load-Balancing facility



Service requests are scheduled according to the load level of each node. The following load levels are used:

**LEVEL0**

A light load. Service requests are usually scheduled to nodes with LEVEL0 or LEVEL1.

**LEVEL1**

A medium load. At rescheduling due to an error, service requests may not be scheduled to nodes with LEVEL1. However, if there are only nodes with LEVEL1 or LEVEL2 at rescheduling, service requests are scheduled to those nodes.

**LEVEL2**

A heavy load. Service requests are usually not scheduled to nodes with LEVEL2. However, if there are only LEVEL2 nodes, service requests are scheduled to LEVEL2 nodes.

The load level of each node is checked at each load check interval. At that time, the

current load level is determined according to the previous load level, the number of queued service requests, the number of remaining service requests, and the server processing rate.

Table 3-12 shows the conditions that determine the load level.

Table 3-12: Conditions that determine the load level

Previous load level	Number of queued service requests: Q	Number of remaining service requests: q	Server processing rate: X	Current load level
LEVEL0	$Q \geq 1$	--	$X < 50$	LEVEL1
LEVEL1	$Q \geq 1$	--	$75 \leq X$	LEVEL0
			$50 \leq X < 75$	LEVEL1
			$X < 50$	LEVEL2
LEVEL2	--	$q = 0$	--	LEVEL0
		$q \geq 1$		LEVEL2

#### Legend

--: Ignored.

Number of queued service requests

Number of service requests that are queued into the schedule queue during a load check interval

Number of remaining service requests

Number of service requests that are remaining in the schedule queue when the load is checked

Server processing rate

Processing rate calculated from the following formula:

Server processing rate = (Number of processed services / (Number of queued service requests + Number of remaining service requests))  $\times$  100

The number of processed services is the number of service requests that are processed during a load check interval.

If the load level is changed, the server information is reported to the name service of each node and the server information is updated. By using the `loadlevel_message` operand in the user service definition, a message reporting the change of the load level can be output.

**(6) Definitions when using internode load-balancing facility**

This section describes the definitions and processing on the TP1/Server Base and TP1/Client sides and RPC processing when using the internode load-balancing facility.

**(a) When the server side determines how to perform load-balancing**

The schedule service of TP1/Server Base distributes loads to nodes that can efficiently process the loads according to the schedule status of each node.

Definition on the server (TP1/Server Base) side

In the definition on the TP1/Server Base side, either:

- include the following settings for the operands in the schedule service definition
 

```
set scd_this_node_first = N (default)
set scd_announce_server_status = Y (default)
```

or

- omit the schedule service definition

Definition on the client (TP1/Client) side

Define `dcscddirect=Y` (for TP1/Client/P) in the client environment definition so that TP1/Client makes a request to the schedule service of TP1/Server Base for load balancing. In the definition on the TP1/Client side, specify where to issue a request for load-balancing for the OpenTP1 node's schedule service TP1/Client.

In this case, OpenTP1 nodes are selected to determine scheduling in the order specified in the `dchost` operand. To select OpenTP1 nodes randomly and not in the order specified in the `dchost` operand, you must add `dchostselect=Y` (for TP1/Client/P) to the definition.

**(b) When the client side determines how to perform load-balancing according to the load information from the server****■ When the client is TP1/Client**

Definition on the server (TP1/Server Base) side

In the definition on the TP1/Server Base side, either:

- include the following settings for the operands in the schedule service definition
 

```
set scd_this_node_first = N (default)
set scd_announce_server_status = Y (default)
```

or

- omit the schedule service definition



### Definition on the client (TP1/Client) side

Define `dccltloadbalance=Y` (for TP1/Client/P) in the client environment definition. By this definition, TP1/Client first determines the OpenTP1 node to which a service request should be issued based on the load level information of each server acquired from TP1/Server Base. Then it performs an RPC. In this case, the information including the load level of each server is retained temporarily. This load level is retained in the area where size is specified in the `dccache` operand (which previously was specified in the `dccltcachetim` operand). That is, the shorter the value in the `dccltcachetime` operand becomes, the newer is the load level of each server that is used to determine the request destination to which an RPC should be issued. At the same time, it must be taken into account that accesses to the name service of TP1/Server Base occur more frequently to acquire the load level information.

#### ■ When the server and client are TP1/Server Base

In the definitions on both server and client sides, either:

- include the following settings for the operands in the schedule service definition
 

```
set scd_this_node_first = N (default)
set scd_announce_server_status = Y (default)
```

or

- omit the schedule service definition

In this configuration, TP1/Server Base already has the load level set for the server depending on when it is going to issue a request. Therefore it performs an RPC for a node with a lower load level. At the time when this request is received, the schedule service does not transfer the request according to the verification of load level but processes it within the local node if possible. The schedule service transfers the request to another node only if the server is blocked or only if the load level of the server in the local node is LEVEL2 and some other node has a server with a lower load level.

#### (c) Operations when internode load-balancing facility is used with other facilities

Table 3-13 shows the operations when the internode load-balancing facility is used with other facilities.

*Table 3-13: Operations of the internode load-balancing facility used with other facilities*

When using	Operation
Permanent connection by TP1/Client	The CUP execution process of TP1/Server Base performs an RPC in the node that established the permanent connection. This is the same operation as in the case when the server and client are TP1/Server Base.
Transaction control API by TP1/Client	The transaction delegated execution process of TP1/Server Base performs an RPC. This is the same operation as in the case when the server and client are TP1/Server Base.
Remote API facility	The RAP-processing server of TP1/Server Base performs an RPC. This is the same operation as in the case when the server and client are TP1/Server Base.

### (7) Extended internode load-balancing facility

The user can specify the following items:

- Rate of schedules made to the LEVEL0 nodes

By specifying the `schedule_rate` operand in the schedule service definition, you can specify the rate of schedules (%) made to the LEVEL0 nodes.

The `schedule_rate` operand is valid only when you specify `Y` for the `DCSCDDIRECT` operand in the client environment definition of TP1/Client to make service requests.

Note that if `Y` is specified for `scd_this_node_first` in the schedule service definition, service requests are scheduled to the local node with priority.

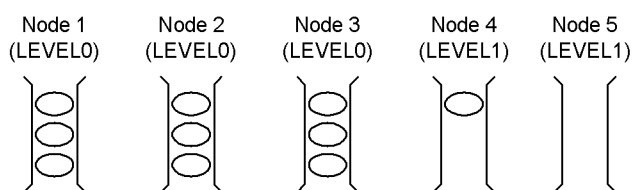
The following example shows how service requests are scheduled when you specify 80 in the `schedule_rate` operand. The number of service requests is 10.

1. The scheduler obtains the load information of all the nodes from the name service and then counts the number of LEVEL0 nodes and the number of LEVEL1 nodes.
2. The scheduler uses the value specified in the `schedule_rate` operand to assign a weight to each number of nodes calculated in step 1. Then the scheduler determines the rate of schedules to be made to the LEVEL0 nodes.  

$$\text{LEVEL0:LEVEL1} = 80 \times 3:20 \times 2 \doteq 85:15$$
3. The scheduler selects a LEVEL0 node using the rate calculated in step 2 and schedules service requests.

Figure 3-51 shows how service requests are scheduled to LEVEL0 nodes.

Figure 3-51: Scheduling service requests to LEVEL0 nodes



#### ■ Load check interval

By specifying the `loadcheck_interval` operand in the user service definition and the user service default definition, you can specify the load check interval for each service group. If the load level is changed when the load is checked, the server information is reported to the name service of each node. Therefore, the server information may be sent out to the network at each load check interval. To prevent this, do not specify a short interval unless it is necessary.

If you do not specify the `loadcheck_interval` operand, the load check interval will be 30 seconds. Whether to check the load is determined every 10 seconds. In other words, a load check is executed on every third check.

When you specify the `loadcheck_interval` operand, the value specified in this operand is the load check interval. This value determines whether the check of the load are executed at the interval that is calculated from the greatest common measure of 10 and the value specified in the `loadcheck_interval` operand for each user server. For example, when you specify 3 for the `loadcheck_interval` operand of SPP1 and 5 for the `loadcheck_interval` operand of SPP2, the interval of the checks is 1 (second) since it is the greatest common measure of 10, 3, and 5. The load check of SPP1 is executed on every third check. The load check of SPP2 is executed on every fifth check.

Therefore, to keep the influence to the system to the minimum, specify a multiple of 5 as the value to be specified for the `loadcheck_interval` operand.

When you specify 0 for the `loadcheck_interval` operand, you can suppress the load check in each service group.

#### ■ Thresholds of load levels

By specifying the `levelup_queue_count` operand and the `leveldown_queue_count` operand in the user service definition and the user service default definition, you can use the numbers of remaining service requests to specify the thresholds that determine the load levels for each service group.

Specify the thresholds as follows:

```
set levelup_queue_count = U1,U2
```

```
set leveledown_queue_count = D0,D1
```

U1: Number of remaining service requests, which determines that the server's load level is upgraded to LEVEL1

U2: Number of remaining service requests, which determines that the server's load level is upgraded to LEVEL2

D0: Number of remaining service requests, which determines that the server's load level is downgraded to LEVEL0

D1: Number of remaining service requests, which determines that the server's load level is downgraded to LEVEL1

The current load level will be determined by the previous load level and the number of remaining service requests.

Table 3-14 shows the correspondence between the number of remaining service requests and the load level.

*Table 3-14: Numbers of remaining service requests and load levels*

Previous load level	Number of remaining service requests: q	Current load level
LEVEL0	$q < U1$	LEVEL0
	$U1 \leq q < U2$	LEVEL1
	$U2 \leq q$	LEVEL2
LEVEL1	$q \leq D0$	LEVEL0
	$D0 < q < U2$	LEVEL1
	$U2 \leq q$	LEVEL2
LEVEL2	$q \leq D0$	LEVEL0
	$D0 < q \leq D1$	LEVEL1
	$D1 < q$	LEVEL2

■ Number of retries to be made if a communication error occurs

If a communication error occurs while service requests are scheduled, the service requests are not usually rescheduled and an error is returned.

However, by specifying the `scd_retry_of_comm_error` operand in the schedule service definition, you can specify the number of schedule retries to be made to nodes other than the failed node.

Note that if the value specified in the `scd_retry_of_comm_error` operand

exceeds the number of nodes with active service groups which are the target of the service requests, the number of nodes with active service groups which are the target of the service requests is the upper limit of retries.

When you specify 0, no retry is made.

To use this function, TP1/Extension 1 must be installed beforehand. If TP1/Extension 1 is not installed, the operation is not guaranteed.

### **(8) Multi-scheduler facility**

In addition to the regular scheduler daemon (called the *master scheduler daemon* hereafter), you can start multiple daemon processes specialized to receive service requests (called the *multi-scheduler daemon* hereafter) to receive several service request messages concurrently. This way, you can avoid the scheduling delay due to reception contention. This solution is called the *multi-scheduler facility*.

To use the multi-scheduler facility, you must specify:

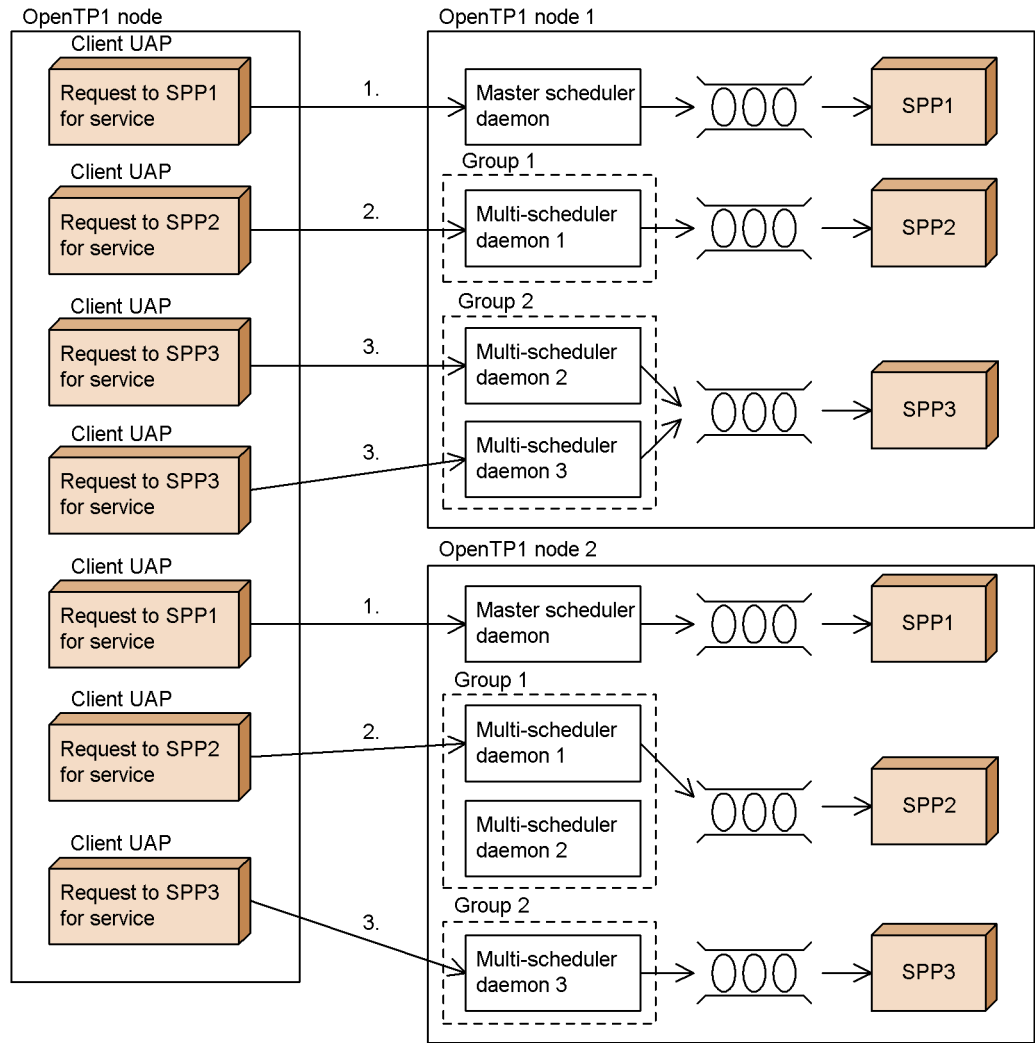
- `scdmulti` in the schedule service definition and `scdmulti` in the user service definition on the RPC receiving side.
- `multi_schedule` in the user service definition on the RPC sending side.

It is also possible to group multi-scheduler daemons by servers that receive requests from a queue. This grouping prevents servers from contending for receiving of service request messages. When multi-scheduler daemons are grouped, you must specify `scdmulti` in the schedule service definition on the server side.

This facility requires TP1/Extension 1 installed. If TP1/Extension 1 is not installed, the operation is not guaranteed.

Figure 3-52 gives an overview of the multi-scheduler facility.

Figure 3-52: Overview of multi-scheduler facility



- Since SPP1 processes a short service request message, OpenTP1 uses the master scheduler daemon to perform scheduling without using the multi-scheduler facility. (See 1. above.)
- Since SPP1 processes long service request messages, OpenTP1 distributes the requests to OpenTP1 nodes 1 and 2, and uses multi-scheduler daemon 1 of group 1 in node 1 or multi-scheduler daemons 1 and 2 of group 1 in node 2 to perform scheduling. (See 2. above.)
- SPP3 processes a short service request message. When there are many service requests, OpenTP1 distributes the requests to OpenTP1 nodes 1 and 2, and uses multi-scheduler daemons 2 and 3 of group 2 in node 1 or multi-scheduler daemon 3 of group 1 in node 2 to perform scheduling. (See 3. above.)

### 3.4.4 Saving shared memory in sharing a buffer area

An area called a *message-storing buffer pool* saves service requests on the shared memory and is secured for each user server. When this area is shared among multiple user servers, the shared memory used by a user server schedule can be saved. If the message-storing buffer tool is shared, specify the following definitions in the concerned nodes.

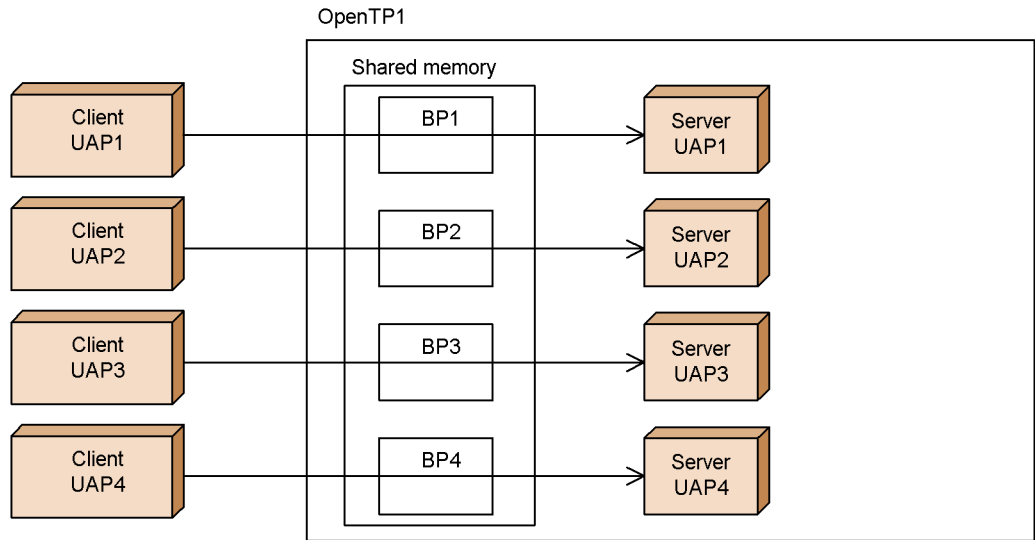
- Specify a user server group name (*schedule buffer group name*) that shares a buffer area in the `scdbufgrp` operand in the schedule service definition.
- Specify a schedule buffer group name using the `scdbufgrp` operand in the user service definitions for every user server to be shared.

If there is no schedule buffer group name specified in the user service definitions, an error will occur when starting the user server.

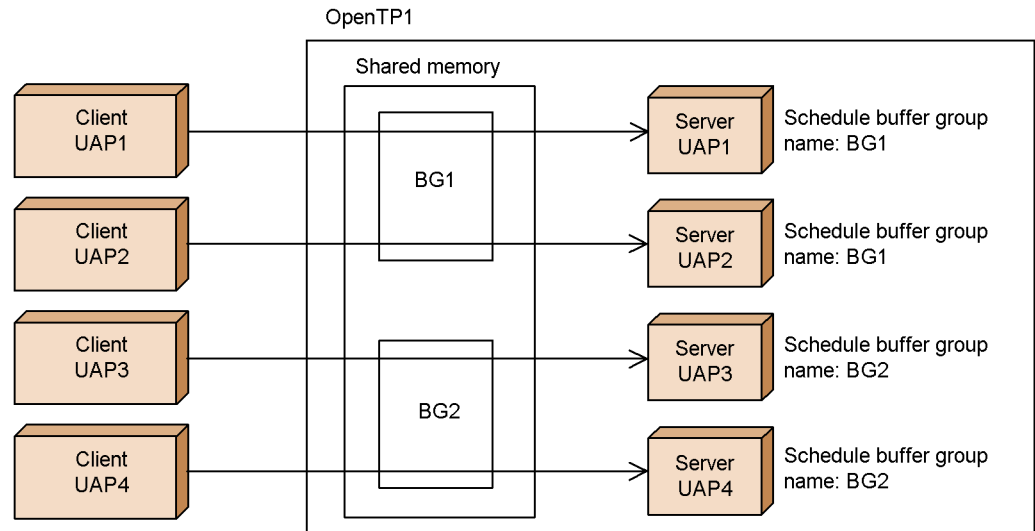
Figure 3-53 illustrates the shared buffer area.

Figure 3-53: Sharing a buffer area

● Exclusive use of a buffer area



● Sharing a buffer area



Legend:

BP: Message-storing buffer pool

BG: Schedule buffer group (specified in the schedule service definition)



**(1) User servers that specify sharing buffer areas**

Specifying the sharing of a buffer area is only valid for a server that receives requests from the queue. Even if the sharing of a buffer area is specified for a server that receives requests from the socket, the specification will be ignored and be executed without sharing the buffer area.

**(2) Notes for sharing a buffer area**

When the processes of the user servers that have sharing specified are started simultaneously, a conflict occurs in the buffer area and schedule performance may decrease. Share the buffer area only if the user servers use non-resident processes, or when it is obvious that the user servers share the buffer simultaneously.

**3.4.5 Example of process control with the Multiserver facility**

This subsection gives an example of how the Multiserver facility controls UAP (SPP or MHP) processes.

For example, suppose that:

- up to six processes can be executed concurrently for the SPP or MHP is six.
- the user service definition has been defined as shown in Table 3-15.

*Table 3-15: Values specified in user service definition*

Items specified in user service definition	Service group G1	Service group G2	Service group G3	Total
Number of resident processes	2	1	1	4
Maximum number of processes in a service group	4	2	2	8
Schedule priority	1	2	3	-

The total of the resident processes for service groups G1, G2, and G3 is four, so two more processes can be activated dynamically before the maximum number (6) of concurrently executable processes is reached. Service groups G1, G2, and G3 use these two processes according to their scheduling priorities. Table 3-16 summarizes what happens, and a more detailed explanation of each step follows the table.

*Table 3-16: Flow of process control*

Steps in procedure	Service group G1	Service group G2	Service group G3
1. Online processing starts.	rr	r	r

### 3. Functions

Steps in procedure	Service group G1	Service group G2	Service group G3
2. G1 and G3 are requested to provide a service.	RR	r	R
3. Service requests for G1 and G3 increase.	RR ■	r	R ■
4. Service requests for G1 increase further.	RR ■ ■	r	R
5. G2 is requested to provide a service.	RR ■ ■	R	R
6. Service requests for G2 increase.	RR ■ ■	R	R
7. Service requests for G1 decrease.	RR ■	R ■	R

#### Legend

r: Resident process standing by (active)

R: Resident process executing a service

■: Non-resident process executing a service

#### 1. Online processing starts.

The resident processes for service groups G1, G2, and G3 are made ready at the start of the online processing. If necessary, two more processes can be started dynamically because the maximum number of executable processes is six but there are only four resident processes.

#### 2. G1 and G3 are requested to provide a service.

The service is executed by the resident processes of G1 and G3.

#### 3. Service requests for G1 and G3 increase.

To handle the service requests, non-resident processes are started for G1 and G3. The services are executed. The number of started processes can be up to the maximum permitted. In this case, since the total number of processes becomes 6, no more processes can be added dynamically.

#### 4. Service requests for G1 increase further.

Because the number of processes has reached the maximum, no more processes can be added dynamically to handle the extra requests for G1.

G1 is provided with another process, however, because G1 has a higher schedule priority than G3. The non-resident process for G3 is terminated when the

currently executing service finishes and a non-resident process for G1 is started.

5. G2 is requested to provide a service.

When a service request is sent to G2, the standing-by service is executed by the resident process of G2.

6. Service requests for G2 increase.

Because the number of processes has reached the maximum, no more processes can be added dynamically to handle the extra requests for G2.

G1 has a higher schedule priority than G2, so the non-resident processes for G1 are not terminated and a non-resident process for G2 is not started.

7. Service requests for G1 decrease.

Decreasing the number of service requests for G1 results in termination of a non-resident process for G1. When the currently executing service is completed for G1, the available non-resident process is started for G2.

Using procedures such as those described above, OpenTP1 can control processes and thereby efficiently process service requests for SPPs and MHPs.

---

## 3.5 OpenTP1 client facility (TP1/Client)

---

This section describes the OpenTP1 client facility (TP1/Client) used by a client WS or PC for requesting services from a server system configured using OpenTP1.

For details about TP1/Client/W and TP1/Client/P, see the manual *OpenTP1 TP1/Client User's Guide TP1/Client/W, TP1/Client/P*. For details about TP1/Client/J, see the manual *OpenTP1 TP1/Client User's Guide TP1/Client/J*.

- TP1/Client communications

TP1/Client can perform the following:

- Remote procedure calls of TP1/Client

A UAP of TP1/Client (CUP), a Java applet, a Java application, or a Java servlet can request a service from an SPP of OpenTP1. When you use TP1/Client/W or TP1/Client/P, you can start a transaction from a CUP and request a service from an SPP.

- Message transmission using the TCP/IP protocol (supported by TP1/Client/W and TP1/Client/P)

Messages can be sent and received between a CUP and an MHP of OpenTP1.

- Communication with XDM/DCCM3

TP1/Client can communicate with old systems such as XDM using the same method used for issuing remote procedure calls from a CUP, a Java applet, a Java application, or a Java servlet to OpenTP1.

- Preparing to request a service from TP1/Client to OpenTP1

When TP1/Client requests a service of OpenTP1, the facility to determine whether or not to accept the request is known as the *User Authentication facility*.

Whether or not the service request is accepted can be set by registering log-in names and passwords for TP1/Client or CUP in the `/etc/passwd` directory of OpenTP1.

The use of the User Authentication facility varies depending on the products.

- For TP1/Client/W and TP1/Client/P for Windows, call the start and end that indicate a function (`dc_clt_cltin_s()` and `dc_clt_cltout_s()` functions) while coding CUP.
- For TP1/Client/P for MS-DOS, executes the TP1/Client commands (`CLTIN` and `CLTOUT` commands) and manages the start and end of the User Authentication facility.

The above preparations are not required when you use TP1/Client/J.

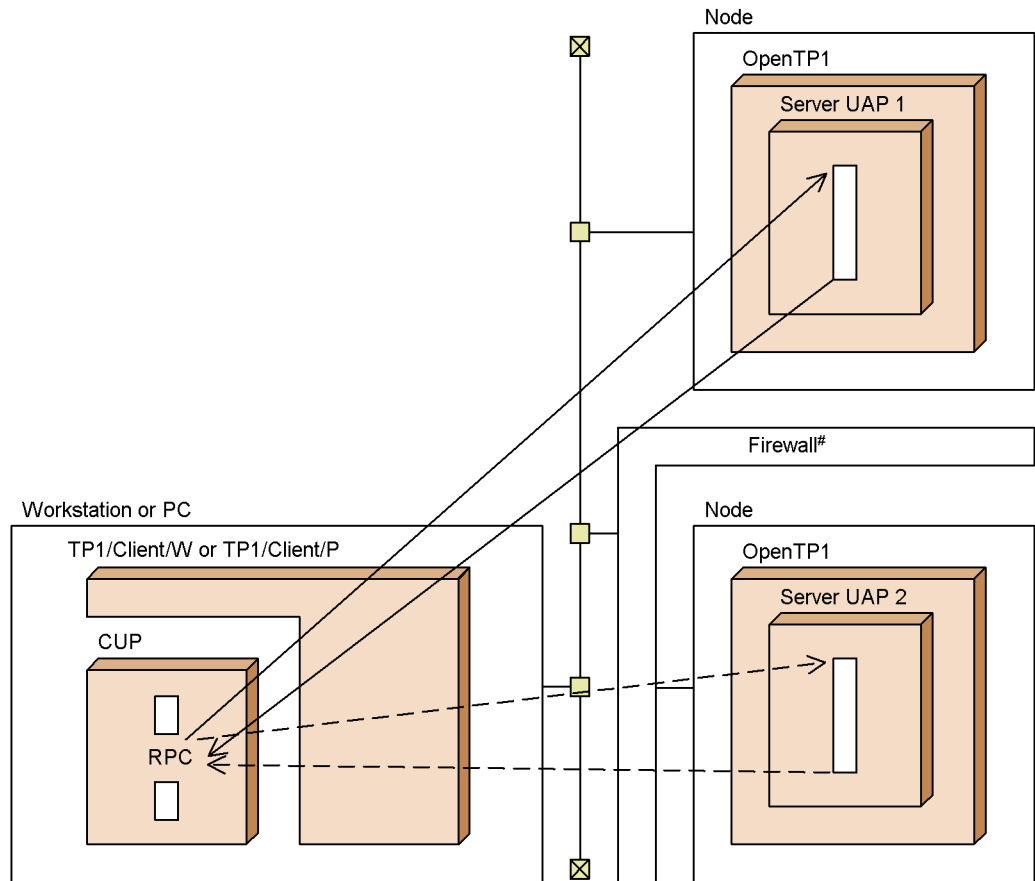
### 3.5.1 Remote procedure calls of TP1/Client

This method allows a CPU to request a service from a server UAP (SPP) of OpenTP1 using library functions of TP1/Client/W or TP1/Client/P.

When you use TP1/Client/W or TP1/Client/P, you can start a transaction from a CUP also.

Figure 3-54 shows the communication between TP1/Client/W or TP1/Client/P and OpenTP1.

Figure 3-54: Communication between TP1/Client/W or TP1/Client/P and OpenTP1



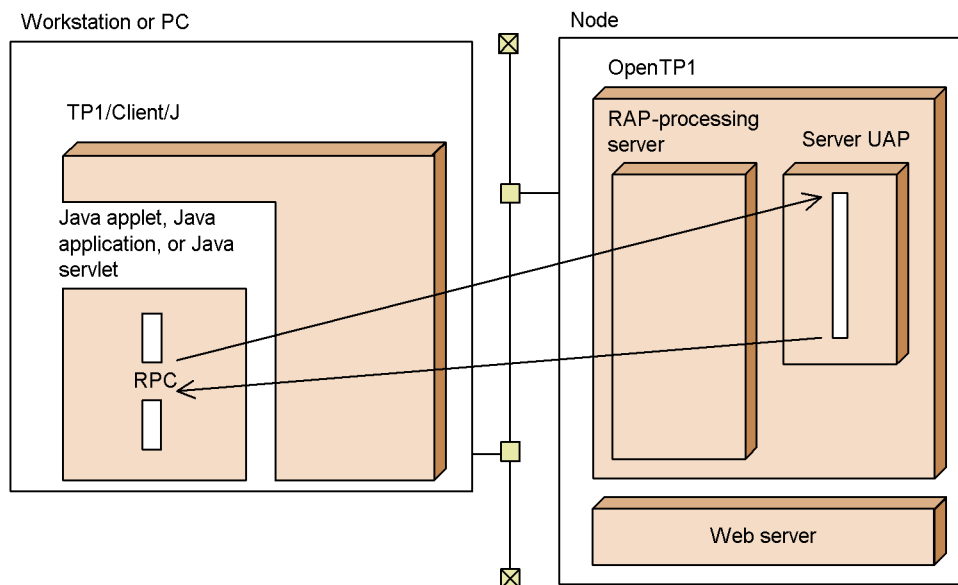
# With an interface for OpenTP1, you can perform communication using a remote procedure call even to a server UAP inside a firewall. However, if a remote procedure call passes through a firewall, the transaction cannot include the server UAP at the destination node.

See the manual *OpenTP1 Version 6 TP1/Client User's Guide TP1/Client/W, TP1/Client/P* for passing through a firewall.

From a Java applet, a Java application, or a Java servlet, you use the class library of TP1/Client/J to request a service from a server UAP (SPP) of OpenTP1.

Figure 3-55 shows the communication between TP1/Client/J and OpenTP1.

Figure 3-55: Communication between TP1/Client/J and OpenTP1



### (1) TP1/Client facilities that need to be defined by OpenTP1

If the following facilities are used in TP1/Client/W or TP1/Client/P, a *client service definition* will be required for an OpenTP1 server.

- Starting a transaction from a CUP

In OpenTP1 that has created the client service definition, a system service is started to manage the transactions started from TP1/Client/W or TP1/Client/P. This system service is called a *client service*. The started client service manages the transactions started by a CUP of TP1/Client.

- Using a permanent connection

In OpenTP1 that has created the client service definition, a system service is started to enable a permanent connection with TP1/Client/W or TP1/Client/P. This system service is called a *client extended service*. The started client extended service establishes a permanent connection with a CUP of TP1/Client.

To use the remote API facility in TP1/Client/W or TP1/Client/P or to have TP1/Client/J request services from an SPP of OpenTP1, OpenTP1 must have the RAP-processing listener service definition.

### (2) Reporting server startup to TP1/Client

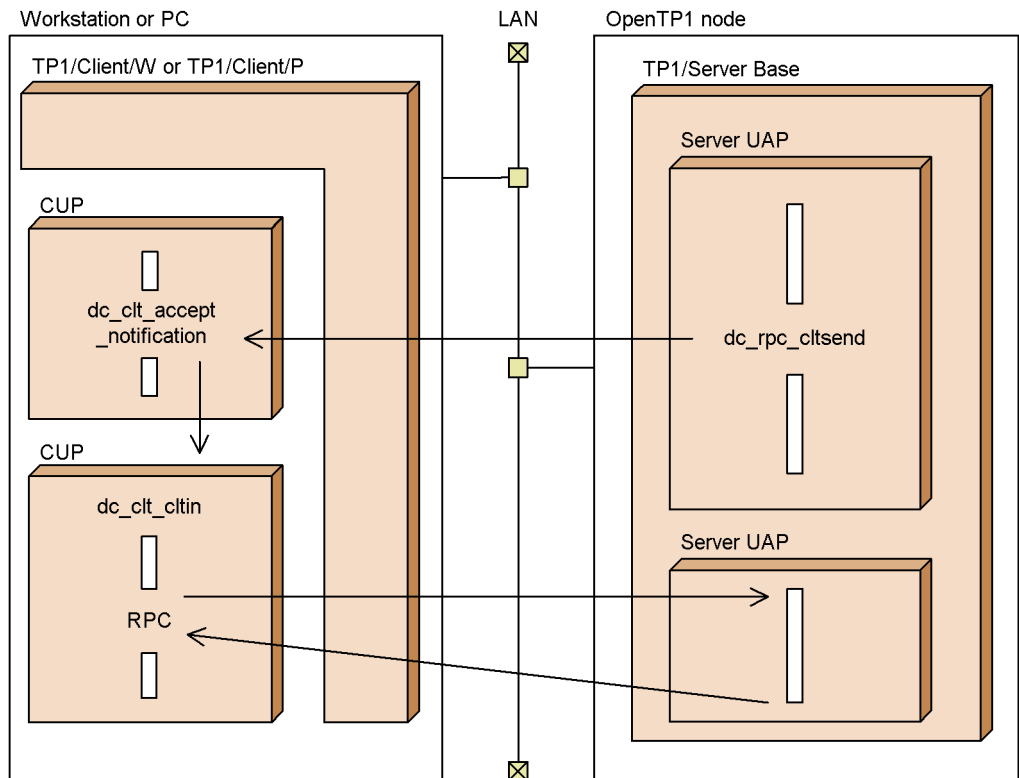
Startup of UAP can be reported from the UAP server of OpenTP1 to the application program (CUP) of TP1/Client/W or TP1/Client/P. The startup of UAP server is

reported by sending data with the `dc_rpc_cltsend()` function to CUP. Using this function can simultaneously report the completion of server startup to the clients.

The data reported by the `dc_rpc_cltsend` function is received by the `dc_clt_chained_accept_notification` or `dc_clt_accept_notification` function of CUP. TP1/ClientW or TP1/ClientP acknowledges that the server is being operated by receiving the data by CUP. After that, a service is requested from CUP to the server. For details about the `dc_clt_chained_accept_notification` and `dc_clt_accept_notification` functions, see the manual *OpenTP1 TP1/Client User's Guide TP1/Client/W, TP1/Client/P*.

Figure 3-56 illustrates communication from a server UAP of OpenTP1 to a CUP of TP1/Client/W or TP1/Client/P.

*Figure 3-56: Communication from a server UAP of OpenTP1 to a CUP of TP1/Client/W or TP1/Client/P*



**(3) Equation for estimating the number of file descriptors used for sockets**

The maximum number of file descriptors used for the sockets in the transactional RPC execution process (`clttrnd`) is specified in the `max_socket_descriptors`



operand of the user service default definition.

Use the following equation to specify the maximum number of file descriptors used for sockets:

$$\uparrow \left( \textit{number-of-UAP-processes-that-involve-communication-by-transactional-RPC-execution-processes} + 1 + \textit{number-of-system-service-processes} \right) / 0.8 \uparrow$$

Legend:

↑ ↑: Rounded up to the nearest whole number.

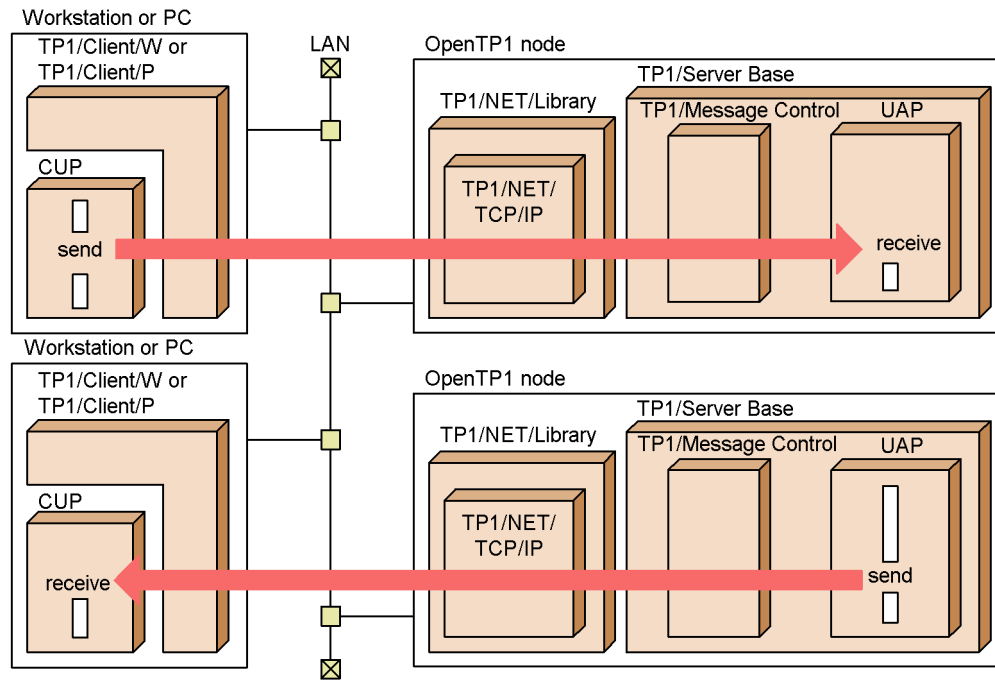
Note that the value you specify here will apply to all UAPs and to some system servers.

### 3.5.2 MCF message exchange using the TCP/IP protocol

In this method, communication with an OpenTP1 UAP is achieved using a message exchange configuration that uses LANs connected with the TCP/IP protocol. Messages can be sent from a CUP of TP1/Client, and messages sent from an OpenTP1 UAP (MHP) can be received via TP1/Client.

When communication is via TCP/IP, the OpenTP1 system from which a service is requested must use TP1/Message Control, TP1/NET/Library, and the TP1/NET/TCP/IP product that corresponds to the TCP/IP protocol. Figure 3-57 illustrates TP1/Client communications using MCF message exchange and the TCP/IP protocol.

Figure 3-57: TP1/Client communications using MCF message exchange and the TCP/IP protocol



### 3.5.3 Communication with XDM/DCCM3

In this method, XDM/DCCM3 can communicate with conventional systems such as XDM in the same way as the issuance of remote procedure calls from CUP to OpenTP1. This section describes how to communicate with the servers other than the OpenTP1 server using a LAN connected by a TCP/IP protocol. You can communicate with XDM/DCCM3 in the same way as for service requests to OpenTP1.

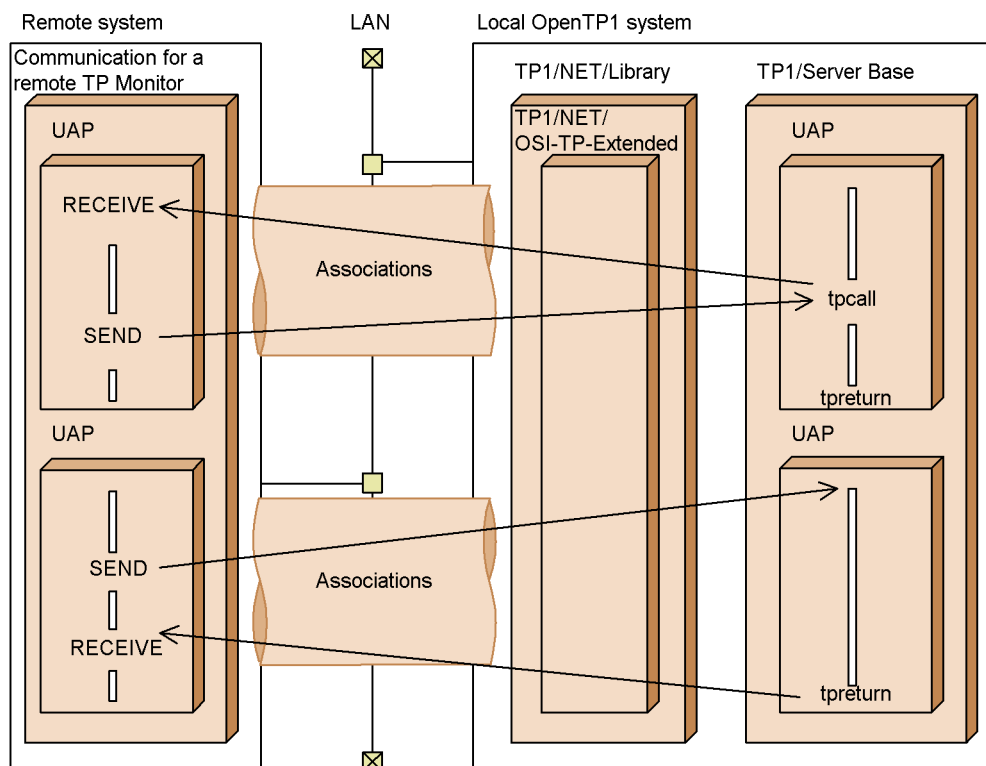
If TP1/Client/W, TP1/Client/P or TP1/Client/J communicates with the XDM/DCCM3, *DCCM3/SERVER/TP1* must be installed in the XDM/DCCM3 host.

### 3.6 Client/server communications using OSI TP

OpenTP1 supports the TCP/IP and OSI TP communication protocols for client/server communications. This section provides an overview of use of OSI TP as the communication protocol. To use the OSI TP communication protocol, you must have TP1/NET/Library, TP1/NET/OSI-TP-Extended, and an OSI TP communication management product. An OpenTP1 system service (*XATMI communication service*) is also required.

The following figure shows the client/server communications mode when OSI TP is used.

Figure 3-58: Client/server communications mode when OSI TP is used



#### 3.6.1 OpenTP1's remote system

OpenTP1's remote system can be an *OpenTP1* system or an *XDM (XDM/DF/TP)* system. When OSI TP communication is used, a transaction started at the local system's OpenTP1 can be extended to the remote system even if the remote system is

not an OpenTP1 system.

APIs with the *XATMI interface* are used for client/server communication under OSI TP. For details about the XATMI interface, see *3.2.4 Communication via RPCs that use the XATMI interface*.

### 3.6.2 Route used for communication

A logical channel is established between the local system's OpenTP1 and the remote system. This channel is called an *association*.# Once an association is established, client/server communications become available.

An association is established between the addresses (PSAP addresses) that represent the contact between the systems.

#

In this manual, a logical channel is referred to as an *association* only when client/server communication is performed using the OSI TP communication protocol.

#### (1) *Calling and called systems*

In OSI TP communications, the attributes of an association are determined by the system that requested establishment of the association. *Calling* refers to issuing an association establishment request, and *called* refers to receiving such a request. The system that issues an association establishment request is the *calling system*, and the system that receives such a request is the *called system*.

To request a service from OpenTP1 for a remote system, an association established by a calling OpenTP1 system is used. Associations established by a called OpenTP1 system are used principally for error recovery communications.

#### (2) *Establishing association*

If you specify in the TP1/NET/OSI-TP-Extended definitions a request for establishment of an association during system startup, an association is established when the system starts. Whether the local system is to be the calling system or the called system depends on the specification of the `nettalccn` protocol-specific definition in the TP1/NET/OSI-TP-Extended definitions.

Make sure that you specify a request to establish association during system startup in the TP1/NET/OSI-TP-Extended definitions.

#### (3) *Releasing association*

When the OpenTP1 system terminates normally, the association is released successfully. If the OpenTP1 system or the remote system terminates abnormally, abnormal release of the association results.

#### (4) *Specifying the system definitions*

The following OpenTP1 definitions are required in order to perform client/server

communications using OSI TP:

- XATMI communication service definition
- Network library definitions (TP1/NET/Library and TP1/NET/OSI-TP-Extended definitions)

OSI TP communications are enabled when correspondence between the XATMI communication service and TP1/NET/OSI-TP-Extended is established by the above definitions.

### 3.6.3 Application programs used for communication

UAPs on OpenTP1 use the XATMI interface for communication with remote systems. Transaction processing can be extended to a remote system. For client/server communications using OSI TP, SUPs and SPPs are supported as UAPs on OpenTP1. Other UAPs on OpenTP1 (such as MHPs) are not supported.

The following describes the XATMI interface communication modes that are supported for client/server communications using OSI TP.

- *Request/response service paradigm* (mode in which responses are received synchronously, mode in which responses are received asynchronously, and mode in which no responses are sent)

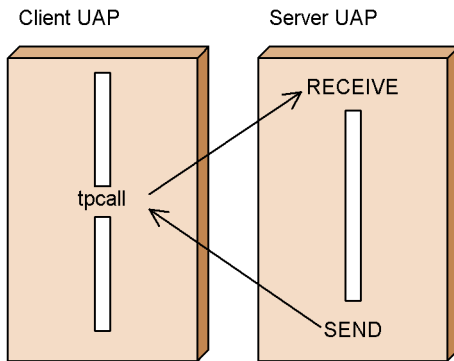
Interactive service communications are not supported in client/server communications using OSI TP.

The following figure shows an application program communication mode when OpenTP1 is the client and XDM/DF/TP is the server.

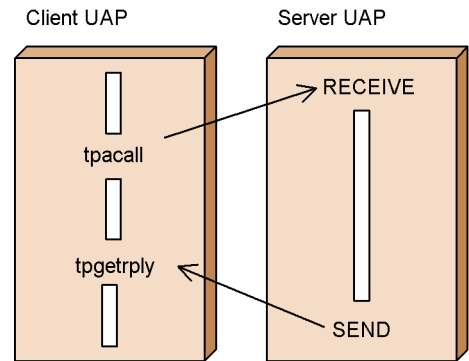
*Figure 3-59: Application program communication mode (when OpenTP1 is the client and XDM/DF/TP is the server)*

● Request/response service communication

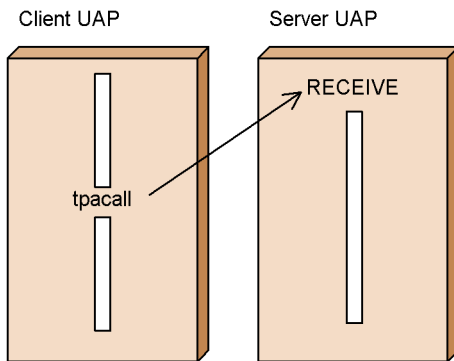
- Synchronous response reception type



- Asynchronous response reception type



- No-response type<sup>#1</sup>



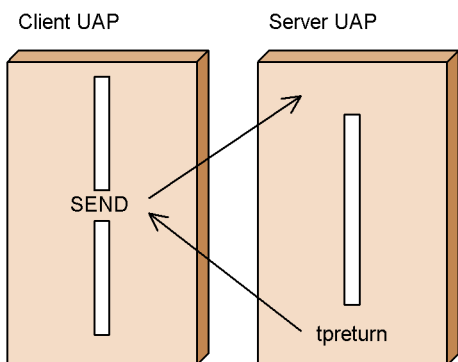
#1: If the no-response type is set, service requests cannot handle transactions.

The following figure shows an application program communication mode when XDM/DF/TP is the client and OpenTP1 is the server.

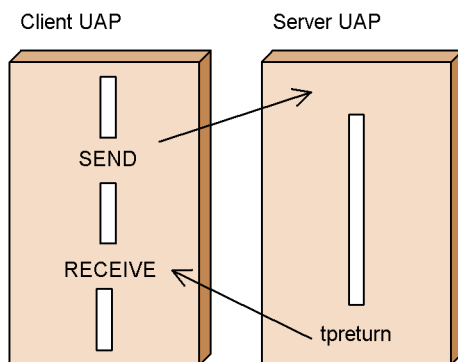
Figure 3-60: Application program communication mode (when XDM/DF/TP is the client and OpenTP1 is the server)

● Request/response service communication

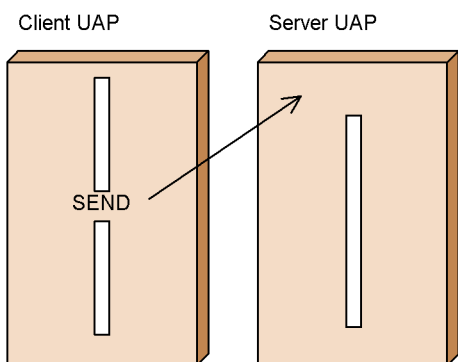
- Synchronous response reception type



- Asynchronous response reception type



- No-response type#1



#1: If the no-response type is set, service requests cannot handle transactions.

**(1) Relationship with transaction processing**

If communication is to be established between two OpenTP1 systems, you can extend transaction processing between the systems. If communication is to be established between an OpenTP1 system and a non-OpenTP1 system, you can extend transaction processing between the systems by using OSI TP.

**(2) SPP for recognizing the association status (communication event processing SPP)**

To perform client/server communications using OSI TP, you must create an SPP for identifying association establishment and release. This is called a *communication event processing SPP*. By creating a communication event processing SPP, you can receive an association release notification event by the SPP. By receiving such a communication event, you can determine the timing of association re-establishment. You use the `dc_xat_connect` function to re-establish association.

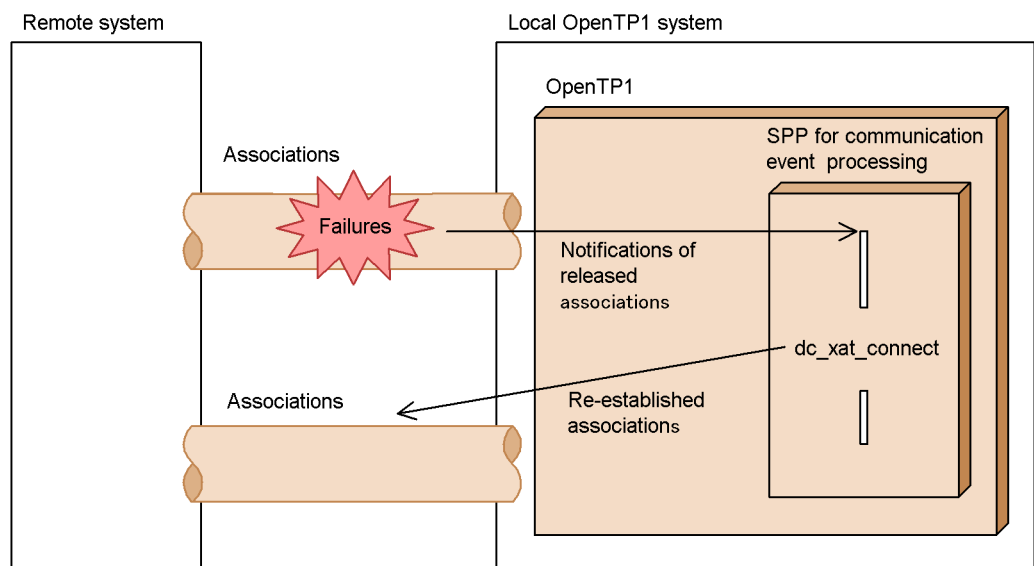
Communication events are always reported, regardless of whether the local system is the calling system or the called system. The communication event processing SPP enables you to determine from the details of a notification event the association's attributes and status.

You must specify in advance in the XATMI communication service definition the service group name and service name of the communication event processing SPP. Communication events are reported on the basis of these specifications.

For details about the communication event processing SPP and the events that can be received, see the *OpenTP1 Programming Guide* and the applicable *OpenTP1 Programming Reference* manual.

The following figure provides an overview of a communication event processing SPP.

Figure 3-61: Overview of a communication event processing SPP





### 3.6.4 Overview of environment setup

This subsection describes how to set up an environment for an OpenTP1 system when OSI TP is used as the client/server communication protocol. The procedures describes in (2) and (3) are the same as for the normal OpenTP1 procedures.

#### (1) *Installing the OpenTP1 products*

Install the following products, which are required for OSI TP communication:

TP1/Server Base, TP1/NET/Library, and TP1/NET/OSI-TP-Extended

#### (2) *Creating the definitions for TP1/Server Base*

After the superuser has chosen the OpenTP1 system administrator, the OpenTP1 system administrator creates the definitions for TP1/Server Base.

#### (3) *Registering OpenTP1*

Execute the `dcsetup` command to register OpenTP1 in the OS.

#### (4) *Creating the definitions for TP1/NET/OSI-TP-Extended*

Create the definitions for TP1/NET/OSI-TP-Extended. Use a utility to analyze the definitions and create the object file, and then store it with the following file name under the directory for storing the OpenTP1 system definitions:

- `$DCCONFPATH/xatcex`

When you use TP1/NET/OSI-TP-Extended in an OpenTP1 system, there is no need to execute the command for registering TP1/NET/Library (`netsetup` command).

For details about how to set up an OpenTP1 system, see *5.1 Setting up an OpenTP1 system*. For details about the prerequisite programs other than OpenTP1, see the manual *OpenTP1 Protocol TP1/NET/OSI-TP-Extended*.

### 3.6.5 In the event of a failure

In the event of a failure in client/server communication using OSI TP, the XATMI interface function that requested the service returns an error. For details about the actual return values, see the notes for the individual functions of the XATMI interface in the applicable *OpenTP1 Programming Reference* manual.

For details about handling communication protocol failures, see the description of error handling procedures in the manual *OpenTP1 Protocol TP1/NET/OSI-TP-Extended*.

---

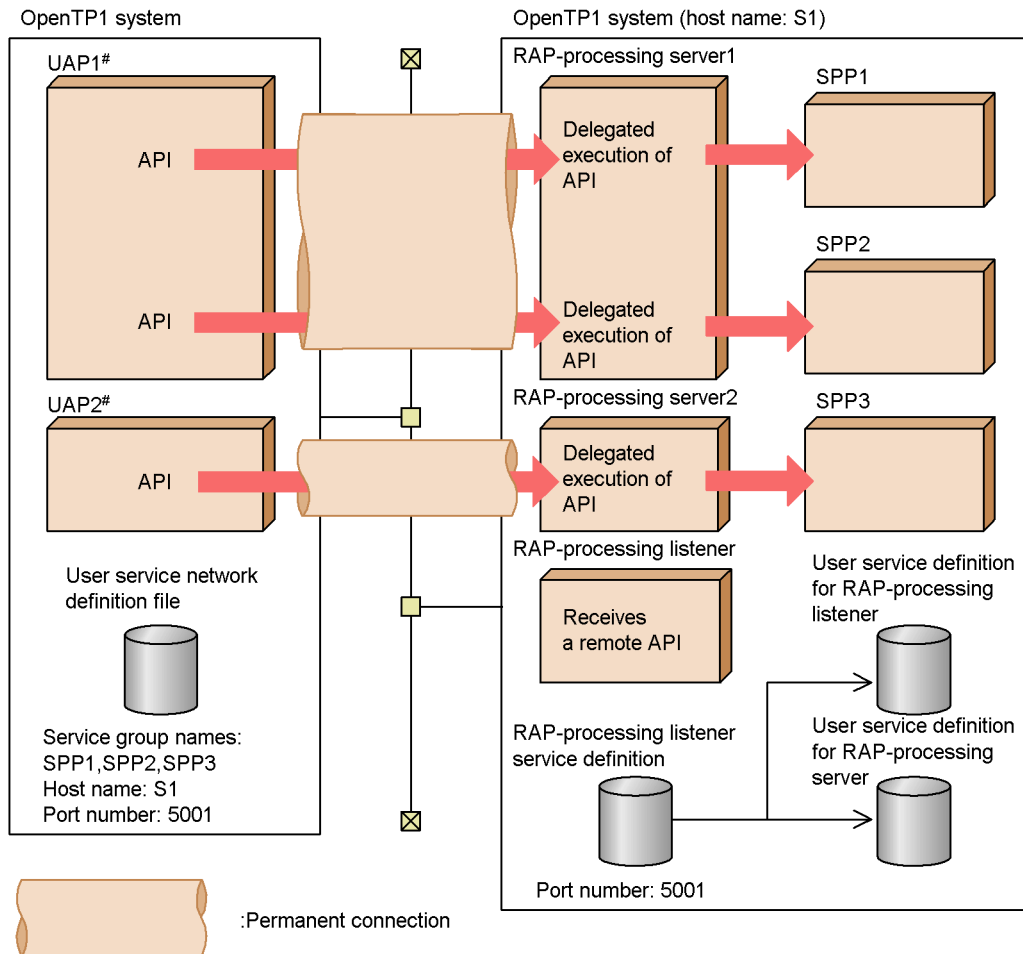
## 3.7 Remote API facility

---

The *remote API facility* is the processing in which OpenTP1 transfers an API issued by the UAP at a client node to the server, and server processes are used to perform the delegated execution of the API. The UAP at the client node that requests the remote API facility is called a *RAP-processing client*. The *RAP-processing listener* of OpenTP1 receives the API issued by the RAP-processing client and the *RAP-processing server* executes the API on the server node. The RAP-processing listener and RAP-processing server operate as user services of OpenTP1.

Figure 3-62 shows an overview of the remote API facility.

Figure 3-62: Remote API facility



Note: Do not use the XATMI interface for the remote API facility. If you use it, the operation of OpenTP1 cannot be assured.

# : For the UAP that uses the remote API facility, specify definition by the `rpc_destination_mode` operand in the user service definition.

Table 3-17 shows the UAPs that can be a RAP-processing client.

Table 3-17: UAPs that can be RAP-processing clients

Program Product	UAPs that can be a RAP-processing client
TP1/Server Base	SUP, SPP, MHP

### 3. Functions

Program Product	UAPs that can be a RAP-processing client
TP1/Client	CUP

The APIs that can be executed remotely for each type of RAP-processing client are listed below.

■ TP1/Server Base and TP1/LiNK acting as the RAP-processing client

C library function	Program for creating a COBOL-UAP
dc_rpc_call	CBLDCRPC ('CALL ')

■ TP1/Client/P and TP1/Client/W acting as the RAP-processing client

C library function	Program for creating a COBOL-UAP
dc_rpc_call_s	CBLDCRPS ('CALL ')
dc_trn_begin_s	CBLDCTRS ('BEGIN ')
dc_trn_chained_commit_s	CBLDCTRS ('C-COMMIT')
dc_trn_chained_rollback_s	CBLDCTRS ('C-ROLL ')
dc_trn_unchained_commit_s	CBLDCTRS ('U-COMMIT')
dc_trn_unchained_rollback_s	CBLDCTRS ('U-ROLL ')

■ TP1/Client/J acting as the RAP-processing client

Method
rpcCall
trnBegin
trnChainedCommit
trnChainedRollback
TrnUnchainedCommit
trnUnchainedRollback

■ TP1/Client for .NET Framework acting as the RAP-processing client

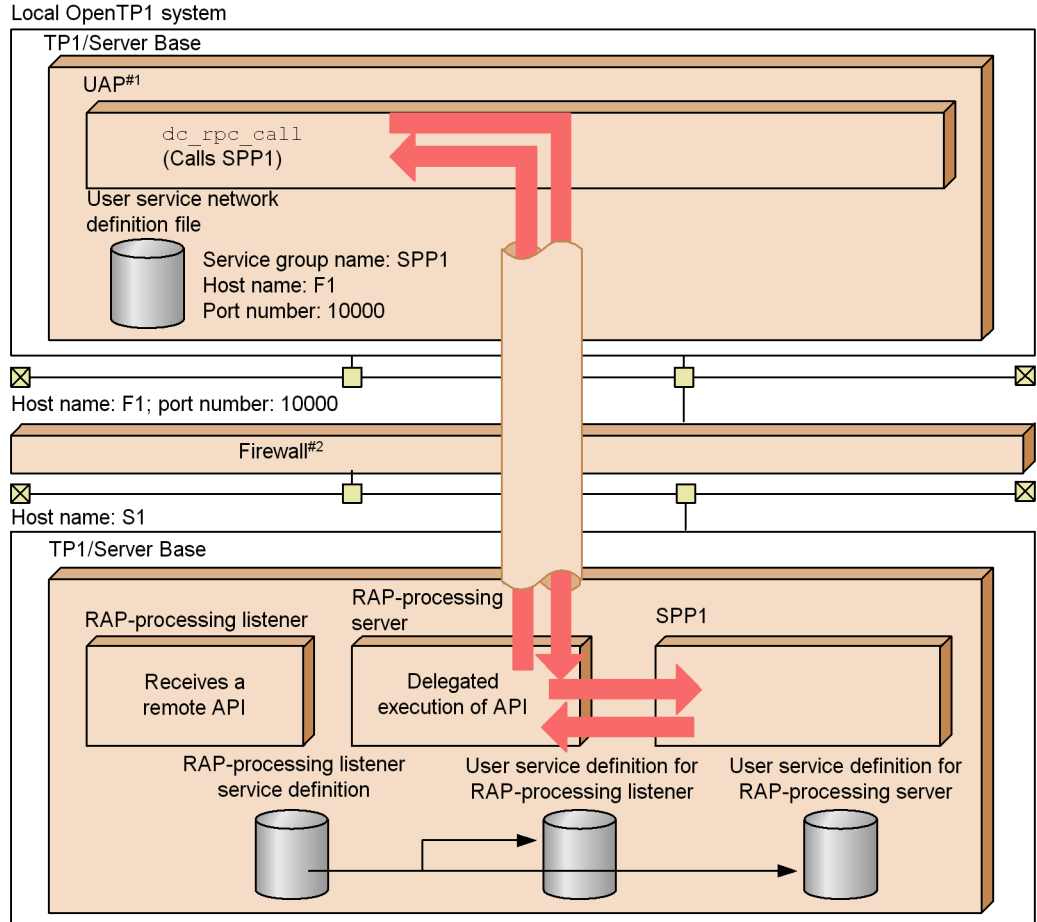
Method
Call

Method
Begin
CommitChained
RollbackChained
Commit
Rollback

### 3.7.1 Example of using the remote API facility

The remote API facility allows requesting a UAP to execute a service even if the UAP is inside a *firewall*. Figure 3-63 illustrates a remote procedure call to inside of a firewall. If you use TP1/Client, see the manual *OpenTP1 TP1/Client User's Guide TP1/Client/W, TP1/Client/P*.

Figure 3-63: Remote procedure call to a UAP inside a firewall



Legend:  : Permanent connection

Note 1: Note the following about passage through a firewall:

- Do not use an asynchronous-response type RPC.
- After an API passes through a firewall, the UAP at the destination (SPP1 in this figure) does not become a transaction branch.

Note 2: Do not use the XATMI interface for the remote API facility. If the XATMI interface is used, correct operation of OpenTP1 is not guaranteed.

#1: For the UAP, specify `definition` in the `rpc_destination_mode` operand in the user service definition.  
 #2: This assumes the use of a Gauntlet firewall.

### 3.7.2 Permanent connection

OpenTP1 establishes a logical communication path (permanent connection) between the UAP that requested a remote API (RAP-processing client) and the RAP-processing server.

Two modes (static connection schedule mode and dynamic connection schedule mode) are available to schedule a permanent connection. You can specify either mode using the `rap_connection_assign_type` operand in the RAP-processing listener service definition.

#### (1) *Static connection schedule mode*

To use the static connection schedule mode, specify `static` in the `rap_connection_assign_type` operand or omit the specification.

The static connection schedule mode allows you to schedule a permanent connection between a RAP-processing client and a RAP-processing server (one to one) by allocating a RAP-processing server when a RAP-processing client requests establishment of a connection. This mode is suitable when the number of RAP-processing clients to be used simultaneously is less than the number of RAP-processing servers to be started (the maximum number of RAP-processing servers is 128 due to an OpenTP1 limit).

When you use the static connection schedule mode, the maximum number of permanent connections that a RAP-processing listener can manage is 256. The maximum number of RAP-processing clients that can be executed with simultaneous permanent connections is the number of RAP-processing servers. Remaining permanent connections cannot be established unless currently executing RAP-processing clients release the permanent connections. Therefore, if the number of RAP-processing clients is greater than the number of RAP-processing servers, perform either of the following:

- Establish permanent connections from RAP-processing clients before you request delegated execution of an API and release the permanent connections from RAP-processing clients after you request delegated execution of an API.
- Start multiple RAP-processing listeners and increase the number of RAP-processing servers that RAP-processing clients establish permanent connections with.

#### (2) *Dynamic connection schedule mode*

To use the dynamic connection schedule mode, specify `dynamic` in the `rap_connection_assign_type` operand.

The dynamic connection schedule mode allows you to dynamically allocate an unused RAP-processing server when a request for delegated execution of an API is issued. This mode does not allocate a RAP-processing server when a RAP-processing client

requests establishment of a connection. This mode is suitable when the number of RAP-processing clients to be used simultaneously is greater than the number of RAP-processing servers but you do not want to increase the number of resources required for RAP-processing servers.

When you use the dynamic connection schedule mode, the maximum number of permanent connections that a RAP-processing listener can manage is 1024. You can specify the maximum number of clients that simultaneously connect to a RAP-processing listener in the `rap_max_client` operand in the RAP-processing listener service definition.

In the dynamic connection schedule mode, a RAP-processing server is allocated only when a request for delegated execution of an API is received from a RAP-processing client and the RAP-processing server is released when the delegated execution of an API is terminated. Therefore, you do not need to establish and release a permanent connection each time you request delegated execution of an API like you do in the static connection schedule mode. Since you do not need to start multiple RAP-processing listeners, you do not need to increase the communication ports for letting packages pass through the firewall. However, note that the response performance might degrade in the dynamic connection schedule mode compared to the static connection schedule mode when the load on the RAP-processing listeners becomes too great.

### 3.7.3 Connection mode

There are two permanent connection management modes (*automatic connection mode* and *non-automatic connection mode*). These modes differ in the method of establishing and releasing the connection. In the automatic connection mode, OpenTP1 manages establishing and releasing a connection. In the non-automatic connection mode, the user manages establishing and releasing a connection. You can specify the connection mode in the user service definition of the RAP-processing client.

#### (1) Automatic connection mode

In this mode, OpenTP1 manages establishing and releasing a permanent connection. A permanent connection is automatically established when the RAP-processing client issues `dc_rpc_call`, specifying a service group name that is defined with the `-w` option in the user service network definition.

A permanent connection starts at issuance of `dc_rpc_call` to a service group in the user service network definition and ends at issuance of `dc_rpc_close` for terminating the remote procedure call.

#### (2) Non-automatic connection mode

In this mode, the user manages establishing and releasing a permanent connection. The user issues the API (`dc_rap_connect` or `dc_rap_disconnect`) for establishing or



releasing a connection at a RAP-processing client. If the user has not established a permanent connection when the RAP-processing client issues `dc_rpc_call` with a service group name defined with the `-w` option in the user service network definition, the `dc_rpc_call` fails.

### 3.7.4 RAP-processing client manager

Even when a RAP-processing client recognizes that a permanent connection has been established, the connection may be invalid (such as when the machine on the RAP-processing server side is powered off). In this case, the RAP-processing client may issue `dc_rpc_call` to send a service request to the RAP-processing server. This can happen even though the service request cannot reach the RAP-processing server, and no reply to the `dc_rpc_call` returns to the RAP-processing client until a timeout occurs.

To prevent this problem, define the RAP-processing client manager service definition and start up the RAP-processing client manager. The RAP-processing client manager determines whether the current connection is valid before the RAP-processing client issues `dc_rpc_call`.

When the RAP-processing client manager determines that the current connection is invalid, it releases the unnecessary connection if it is in the automatic connection mode. Then it establishes a connection again, and issues `dc_rpc_call` to send a service request. If in the non-automatic connection mode, the `dc_rpc_call` returns an error due to `DCRPCER_NET_DOWN`.

To set the operating environment of the RAP-processing client manager, execute the `rapsetup` command. To start up the RAP-processing client manager, execute the `rapdfgen` command and create a user service definition for the RAP-processing client manager.

Note:

When the RAP-processing client manager is restarted after abnormal termination or when the RAP-processing client manager has terminated normally, a RAP-processing client may forcibly close the current connection, assuming the connection to be invalid. If the forcibly closed connection was valid, OpenTP1 outputs message KFCA26965-E (when the static connection schedule facility is used) or message KFCA26956-W (when the dynamic connection schedule facility is used).

For details on messages KFCA26965-E and KFCA26956-W, see the manual *OpenTP1 Messages*.

### 3.7.5 Definitions necessary for using the remote API facility

The following definitions need to be set to use the remote API facility between TP1/Server Bases.

### 3. Functions

- On a RAP-processing client side
  - User service definition
  - User service network definition
  - RAP-processing client manager service definition(optional)
- On a RAP-processing server side
  - RAP-processing listener service definition

#### **3.7.6 Prerequisites for using the XA resource service**

To use the XA resource service, the remote API facility and the following program products are required:

When linking with a J2EE application server

- TP1/Client/J
- uCosminexus TP1 Connector or Cosminexus TP1 Connector

When linking with a .NET Framework application

- Client .NET
- Connector .NET

XA resource service can use either of two methods of scheduling a permanent connection: static connection schedule mode or dynamic connection schedule mode.

For details about the XA resource service, see *3.1.7 Transaction control based on the XA resource service*.

---

## 3.8 Dynamic loading of service functions

---

The service functions in the UAP shared library can be loaded (read) dynamically. Service functions can be added or deleted from an SPP simply by changing the definition. The UAP shared library is created by linking the set of UAP object files generated by compiling the UAP source files.

When you load service functions dynamically, you can add or delete service functions simply by changing the `service` operand of the user service definition without having to change the stub or re-create the UAP's executable file. Because the service functions are loaded when the UAP starts, no stub or service functions are needed in order to create the UAP's executable file. Also, because no RPC interface definition is required, there is no need to re-create the UAP's executable file when adding a new service to or deleting a service from an existing UAP.

Because dynamic loading of service functions saves work, it is particularly useful in a system where services are frequently added or deleted.

Note

Dynamic loading of service functions is applicable only to SPPs and MHPs, except in the following cases:

- The user server uses the XATMI interface.
- The TP1/Offline Tester is being used.

### 3.8.1 Examples of using dynamic loading of service functions

Dynamic loading of service functions can be used on its own, or in combination with a stub.

#### **(1) Examples of using dynamic loading of service functions on its own**

On receipt of a service request from a client UAP, the server acquires the service functions from the UAP shared library and executes them.

The following figures show examples of UAPs (SPP and MHP) that use dynamic loading of service functions.

Figure 3-64: Example of a UAP (SPP) that uses dynamic loading of service functions

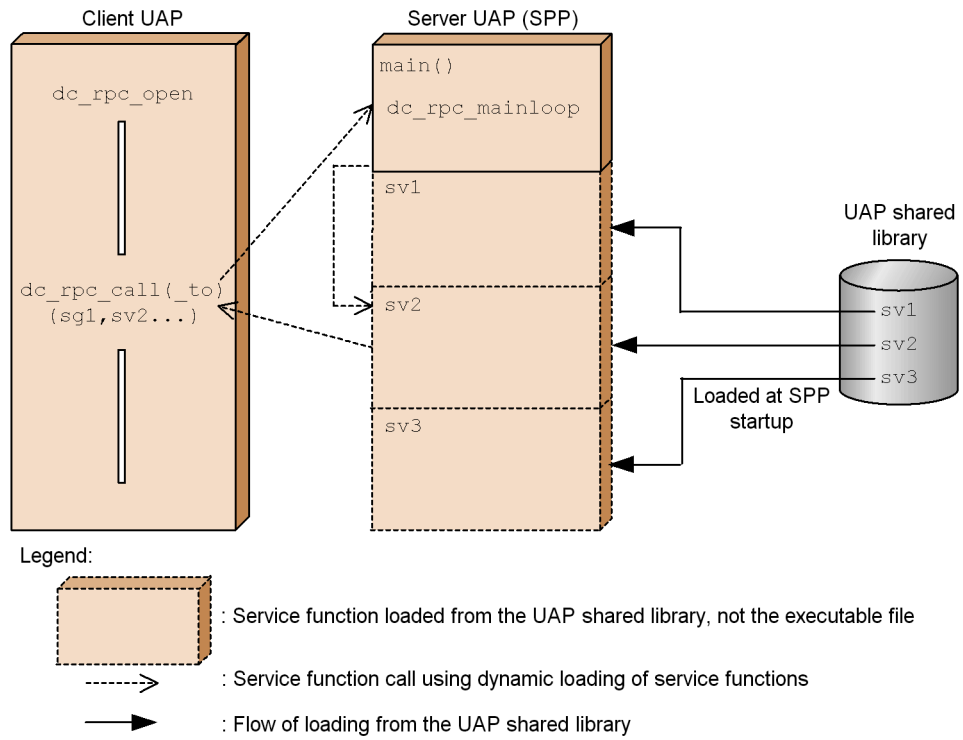
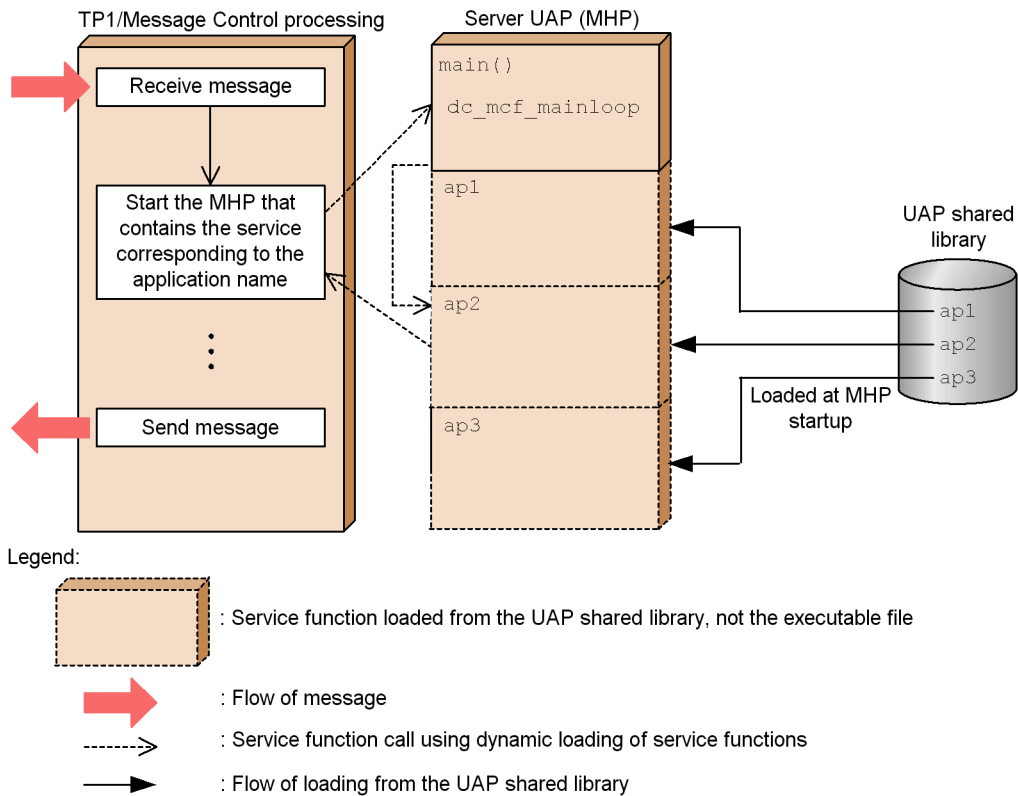


Figure 3-65: Example of a UAP (MHP) that uses dynamic loading of service functions

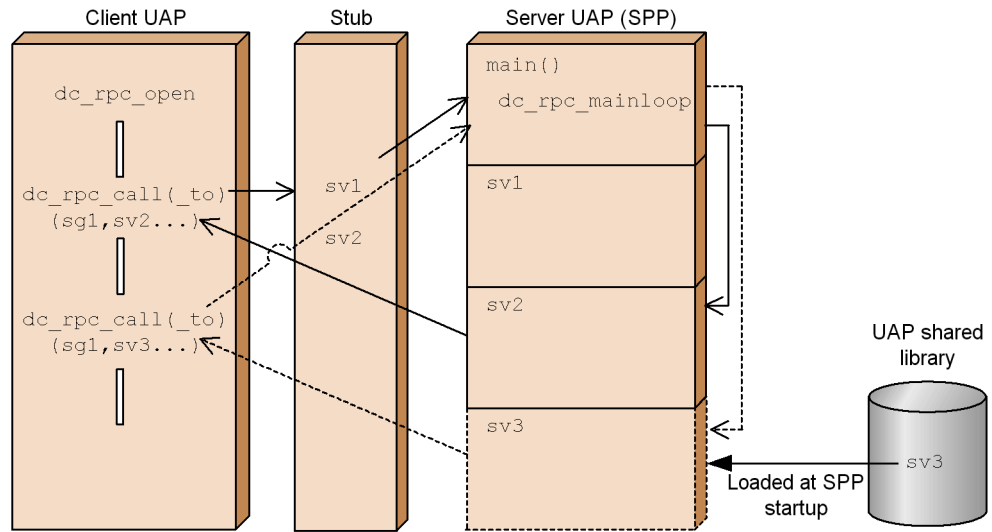


**(2) Examples of using dynamic loading of service functions with a stub**

Dynamic loading of service functions can be used in combination with an SPP that uses a stub and service functions. When they are used together and services are added or deleted, there is no need to re-create the executable file for the UAP that uses a stub and service functions.

The following figures show examples of UAPs (SPP and MHP) that use dynamic loading of service functions in combination with a stub.

*Figure 3-66: Example of a UAP (SPP) that uses dynamic loading of service functions*



Legend:





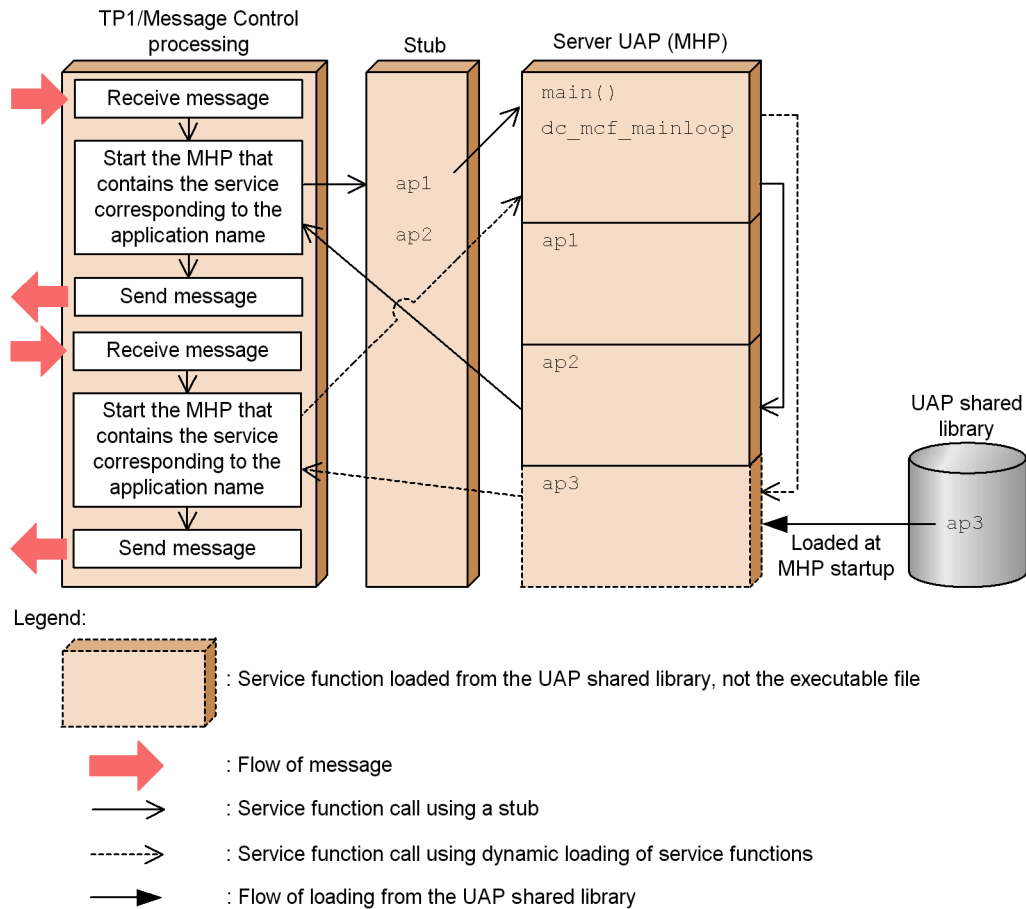
-  : Service function loaded from the UAP shared library, not the executable file
-  : Service function call using a stub
-  : Service function call using dynamic loading of service functions
-  : Flow of loading from the UAP shared library

Figure 3-67: Example of a UAP (MHP) that uses dynamic loading of service functions and a stub



### 3.8.2 Preparation required for using dynamic loading of service functions

The following describes the preparation required for using dynamic loading of service functions.

#### (1) Setting procedure

To set up dynamic loading of service functions:

1. Create a UAP shared library from the service functions.
2. In the `service` operand of the user service definition, specify the service names, entry point names, and UAP shared library name.

For details about the `service` operand, see the manual *OpenTPI System Definition*.

**(2) Criteria for relinking a UAP object**

Table 3-18 describes whether the UAP object needs to be relinked after a service function is added.

*Table 3-18: Criteria for relinking UAP objects*

No.	UAP object in which a service is added or deleted	Service added by	Relink
1	UAP object that uses only the RPC interface definition	Using the RPC interface definition	Yes
		Using dynamic loading of service functions	No
2	UAP object that uses only dynamic loading of service functions	Using the RPC interface definition	Yes
		Using dynamic loading of service functions	No
3	UAP object that uses both the RPC interface definition and dynamic loading of service functions	Using the RPC interface definition	Yes
		Using dynamic loading of service functions	No

Legend:

Yes: The UAP object must be relinked.

No: The UAP object does not need to be relinked.



## 3.9 Additional Features

In addition to the system services described previously in the manual, OpenTP1 provides several other service facilities.

To use the resource manager, install the TP1/Resource Manager Monitor in the OpenTP1 system. Otherwise, use the standard facilities provided by TP1/Server Base.

### 3.9.1 Locking resources

OpenTP1 provides locking facilities via a resource manager that maintains the consistency of each resource so that more than one user can share not only DAM and TAM files but also other resources.

#### (1) Management units and scope of a lock

OpenTP1 manages a lock from the time a lock request is issued until the time the lock is released. A lock is usually managed for each *global transaction*; however, the DAM-service locks are managed for each transaction branch.

The scope of a lock is within one OpenTP1 system. A UAP on one OpenTP1 system cannot lock a resource on another OpenTP1 system. Even in one OpenTP1 system that is composed of multiple nodes, a lock between nodes is not possible.

#### (2) Requesting a resource lock with the PR or EX lock mode

The `dc_lck_get` function can be called from a UAP to lock a resource. In the arguments, specify the resource name and lock mode (method of preventing other UAPs from accessing the resource).

Two lock modes are available: *PR mode* excludes only UAPs that update the resource, allowing access by other UAPs that only reference the resource. *EX mode* grants exclusive use of the resource, prohibiting access by all other UAPs.

The following table lists and describes the lock modes.

Table 3-19: Lock modes

Lock mode	Type	Description
PR	Reference	While a UAP is using the resource, OpenTP1 permits other UAPs to access the resource for reference only, and prohibits other UAPs from updating the resource.
EX	Update	While a UAP is using the resource, OpenTP1 prohibits all kinds of access from other UAPs.

Note

PR: Protected Retrieve

EX: EXclusive

The following table lists and describes when resources can be shared depending on the combination of lock modes.

*Table 3-20:* Possibility of sharing resources depending on combination of lock modes

Lock mode of UAP currently using the resource	UAP that requests access with PR mode	UAP that requests access with EX mode
PR mode	Can share	Cannot share
EX mode	Cannot share	Cannot share

### (3) Releasing a lock

A lock on a resource can be released by a UAP issuing:

- a lock-release request for each resource (e.g., by issuing the function `dc_lck_release_byname()`)
- a global lock-release request (e.g., by issuing the function `dc_lck_release_all()`)

OpenTP1 automatically releases all the resources from locks at the termination of a transaction. This ensures that resources are never left monopolized even if the user omits issuing the release request or the UAP terminates abnormally.

### (4) Waiting to use a resource

In a parameter of the lock-request function `dc_lck_get()`, you can specify whether or not the UAP should wait if it cannot immediately use a resource. A UAP cannot use a resource when:

- another UAP is already using the resource and has locked the resource in the EX mode
- the UAP attempts to use the resource with a lock in the EX mode while another UAP is already using the resource and has locked the resource with the PR mode

If the parameter specifies that a UAP is to wait until a resource can be locked, OpenTP1 makes the UAP that wants to use a resource wait until the lock on the resource is released.

If the parameter specifies that a UAP is not to wait until a resource can be locked, an error is immediately returned.

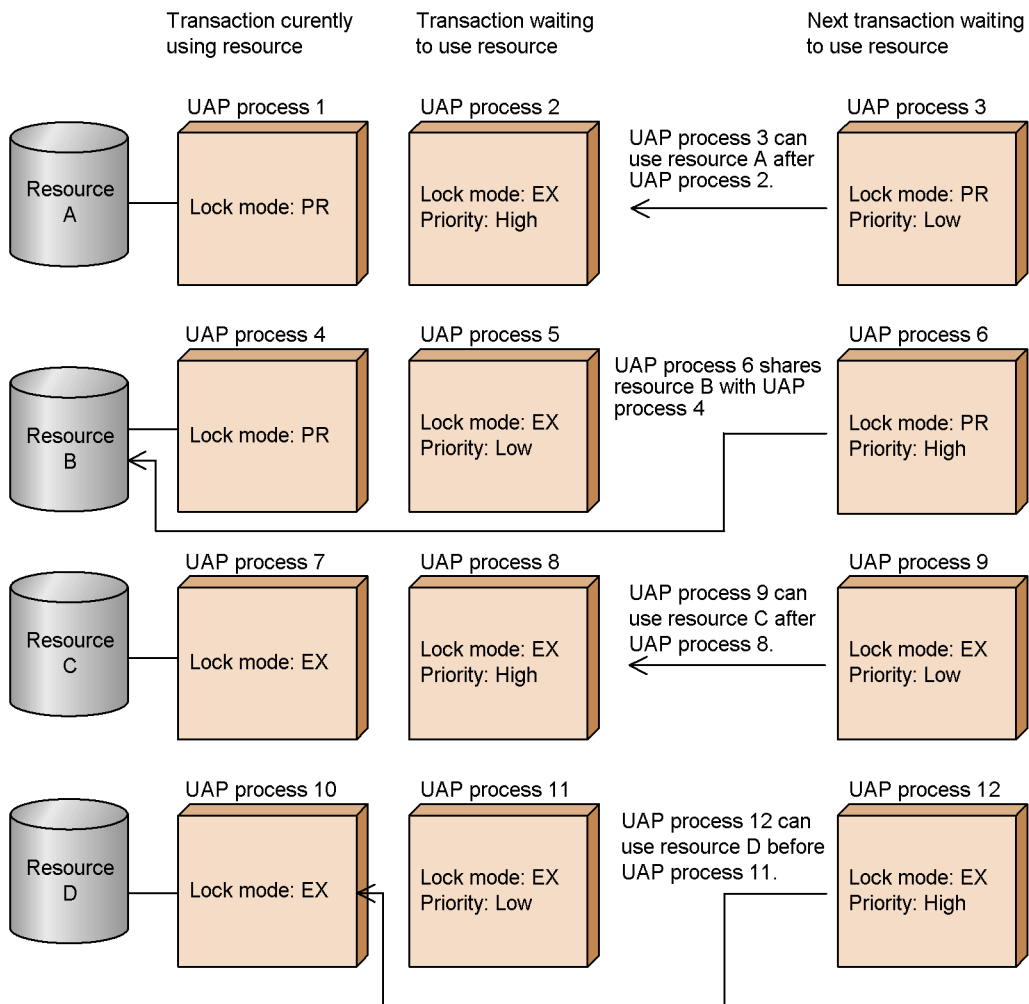
When several UAPs want to use a resource, the lock-wait priority determines which UAP is to next use the resource. The lock-wait priority for a UAP is specified in the user service definition for each UAP service group. A UAP with a higher priority might be able to use a resource earlier than a UAP with a lower priority. The fact that

a UAP with a low priority might access a resource after UAPs with higher priorities needs to be kept in mind when assigning priorities.

In the `lck_wait_timeout` operand in the lock service definition, you can specify how long a UAP should wait for a resource lock to be released. If a lock is not released before this time expires, a lock-request error is returned.

Figure 3-68 illustrates how lock-wait priority affects the sequence in which UAPs can use a resource.

Figure 3-68: Priorities determining sequence in which resources are used

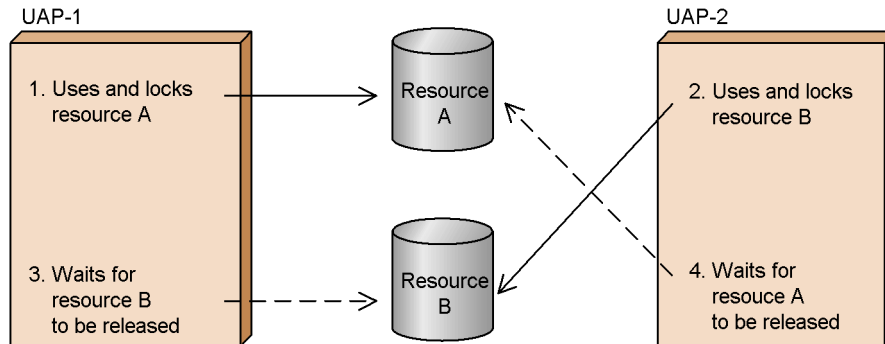


**(5) Deadlocks in TAM and DAM files**

This section describes how TAM and DAM lock functions are affected by deadlocks.

When two or more UAPs lock multiple resources in different sequences, an undesirable situation called a *deadlock* might occur. For example, in one form of deadlock, one UAP might lock resource A and wait for resource B to be released while another UAP might be locking resource B and waiting for resource A to be released. Figure 3-69 illustrates a deadlock.

Figure 3-69: Example of a deadlock



OpenTP1 detects deadlocks differently depending on whether the UAPs are on the same or different nodes. When UAPs are running on the same node, OpenTP1 automatically checks at regular intervals to detect deadlocks. When UAPs are running on different nodes, however, OpenTP1 cannot directly detect a deadlock so OpenTP1 checks whether any UAP has exceeded the specified time-limit for waiting for a resource. In the `lck_wait_timeout` operand in the lock service definition, you can specify the time-limit for waiting for a resource.

**(a) Handling deadlocks**

When a deadlock occurs, OpenTP1 returns an error to the lock request from the UAP process that has the lowest lock-wait priority. You can specify a UAP lock-wait priority with the `deadlock_priority` operand in the user service definition.

When a deadlock causes an error to be returned by the function that attempted to reserve the resource, you should do the following in the UAPs:

- For SUPs or SPPs

When a deadlock occurs during SUP or SPP processing, you should roll back the transaction by using a rollback function (e.g., `dc_trn_unchained_rollback()`). SUPs or SPPs that have been rolled back because of a deadlock are not re-executed. You should ensure that the client UAP re-requests the relevant service.

- For MHPs

When a deadlock occurs during MHP processing, issue the `dc_mcf_rollback()` function to roll back. In this function you can specify whether to re-execute the rolled back application.

### (b) Output of deadlock information and timeout information

When a deadlock occurs, detailed *deadlock information* about the UAPs that caused the deadlock can be output to a directory in a node that contains the lock service.

When a UAP waiting for the release of a resource exceeds the time specified in the `lck_wait_timeout` operand in the lock service definition, the function issued by the UAP returns an error. In such a case, detailed *timeout information* about the resource that the UAP wanted to reserve is output to a directory in the node that contains the lock service.

In the `lck_deadlock_info` operand in the lock service definition, you can specify whether to output deadlock information and timeout information. For details about the format of deadlock and timeout information, see the *OpenTP1 Programming Guide*.

There are two ways to delete the acquired deadlock information:

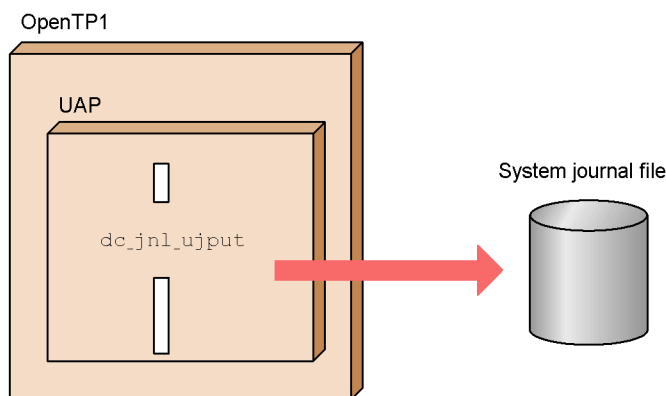
- By executing the `lckrminf` command.

By deleting, at the start of OpenTP1, information that was created up to the previous online processing: Specify the delete conditions in the `lck_deadlock_info_remove_level` operand of the lock service definition.

## 3.9.2 Acquisition of a user journal

Using the functionality for collecting historical information during UAP processing, you can acquire selected information into a system journal file. Figure 3-70 illustrates how a *user journal* (UJ) is acquired.

Figure 3-70: Acquisition of a user journal



For details about how to collect a UJ, see the *OpenTP1 Programming Guide*.

### 3.9.3 Journal maintenance facilities

The journal maintenance facilities are for editing the journal data copied to an unload journal file. For example, you can merge multiple unload journal files into one. The commands are:

`jnlcopy`

For duplicating journals.

`jnlcolc`

For integrating file recovery journals.

`jnledit`

For editing journals.

`jnlrput`

For outputting unloaded-journals file records.

This section explains these commands.

#### (1) *Duplicating journals*

You can use the `jnlcopy` command to duplicate journals. The command provides a record duplicating facility and a merge facility.

The *record duplicating facility* outputs a specified range of unloaded-journals file information to the standard output. Information is output in units of records and the output range is specified in `jnlcopy` as the date the journal was obtained.

Also, an unloaded-journals file for each type of journal can be copied by specifying the journal type (e.g., AJ or BJ) in the `jnlcopy` command.

The *merge facility* merges multiple unloaded-journals files into one file. Only those unloaded-journals files that are output to the same online system and have consecutive generation numbers can be merged.

The facilities provided by the `jnlcopy` command are referred to as the *journal duplicating facilities*.

#### (2) *Integrating file recovery journals*

You can use the `jnlcolc` command to extract only the data necessary for DAM FRC or TAM FRC from the unloaded-journals files, and output the data to the standard output. The data necessary for file recovery in the unloaded-journals files are integrated into a file at one time. This file is called an *integrated journal file*.

A DAM or TAM file in which an error occurred is recovered from the backup file and unloaded-journals files. OpenTP1 itself integrates the unloaded-journals files and

performs the recovery. At file recovery, if the newest integrated journal file created by the user is available, the recovery time can be reduced.

An unloaded-journals file includes history information that is used to recover undetermined transactions. This history information is stored in an *inheritance file* different from the integrated journal so that the information can be inherited at the next journal integration. The data necessary for file recovery is integrated at the next journal integration from the inheritance file and unloaded-journals files.

The facilities provided by the `jnlcolc` command are referred to as the *file recovery journal integrating facilities*.

### **(3) Editing journals**

You can use the `jnledit` command to output the contents of unloaded-journals files to the standard output in a listing for each file. Information about attributes of one specified journal can also be output.

You can extract required data from all the journals by specifying, in the `jnledit` command, the range of data in journal records or journal blocks. Journal records can be checked as hexadecimal numbers or as hexadecimal numbers and characters.

You can output unedited user journal records contained in an unloaded-journals file to the standard output.

The facilities provided by the `jnledit` command are referred to as the *journal editing facilities*.

For details about the journal editing facilities, see the manual *OpenTP1 Operation*.

### **(4) Outputting unloaded-journals file records**

You can use the `jnlrput` command to output unloaded-journals file records. The records can be output to the standard output as they are without editing the user journal record information or CPU use time information of the transaction branch in the unloaded-journals file.

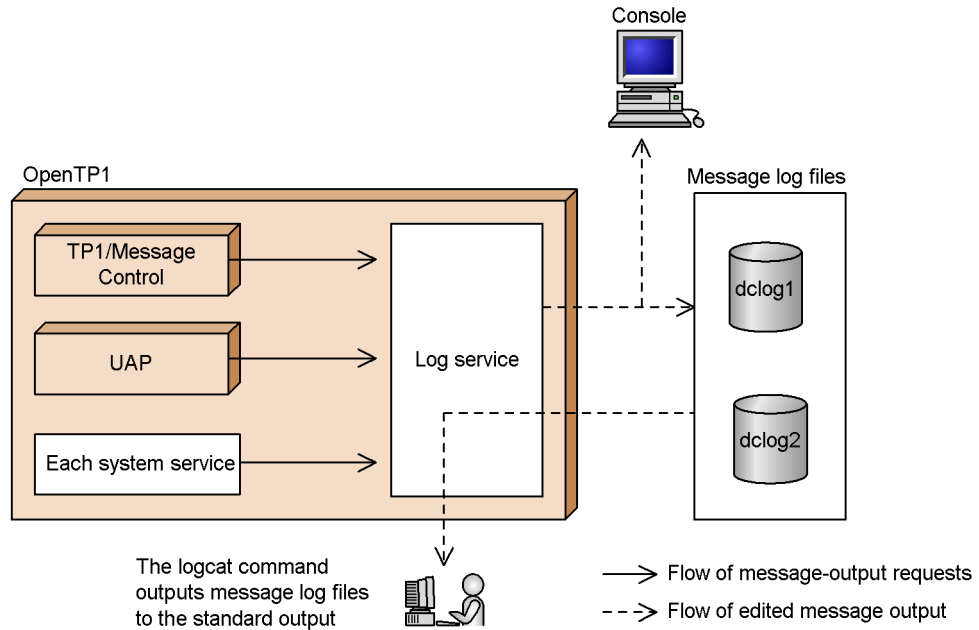
The information to be output can be selected by specifying the items such as the range, journal type, and journal collection mode.

## **3.9.4 Obtaining the message log**

OpenTP1 enables the editing and storing of system messages, and thereby enables the monitoring of online systems.

Figure 3-71 gives an overview of how the message log is obtained.

Figure 3-71: Obtaining a message log



For details about message-log operations, see the manual *OpenTP1 Operation*.

### (1) The OpenTP1 log service and message log files

The OpenTP1 *log service* manages system messages. For example, the log service:

- receives message-output requests from each system service, MCF, and from UAPs
- edits the messages
- outputs the messages to a *message log file*

The log service outputs a message to one of two message log files: `dclog1` or `dclog2`. Using round-robin scheduling, the log service outputs messages to one of the files and when this file is full, the log service reports this fact and proceeds to output messages to the other file. The previous generation of the message log is preserved. When the second message log file is full, the log service reports this fact and proceeds to overwrite the messages in the first file. Messages that the log service outputs to a message log file are simultaneously output to the console.

You can use the command `logcat` to output system messages from a message log file to the standard output. OpenTP1 compares the last update times of `dclog1` and `dclog2` and then outputs the information in both message log files in chronological



order.

### **(2) Requesting numbered messages**

You can request that OpenTP1 assign *message sequence numbers* to each message in the message log files and this information can be obtained as additional message-log information. Even if an error removes a message, the missing sequence number indicates that the message is missing.

You can use options of the command `logcat` to specify how the message log is to be formatted.

### **(3) Suppressing the message log**

If a lock error occurs, OpenTP1 outputs the message log. Repeated attempts to re-execute a UAP that returned an error results in increasing the size of the message log. For the DAM files, you can specify in the `dam_message_level` operand in the DAM service definition whether to output the messages returned by lock error. If you omit this specification, no message is output.

### **(4) Specifying the language for the output message log**

You can choose whether to output the message log in English or in Japanese. Specify the required language in the `LANG` environment variable in `putenv` format in the system common definition. If you omit this specification, the message log is output in English.

## **3.9.5 Reporting a message log**

The OpenTP1 message log can be reported to an exclusively created application program in a system. The application program that receives a report can report the OpenTP1 status to other vendors' application programs.

To report a message log, specify `Y` in the `log_notify_out` operand in the log service definition of OpenTP1.

### **(1) Application programs that can receive a message log report**

Only application programs created for reception can receive a message log report. The UAPs (SUP, SPP, or MHP) of OpenTP1 cannot receive this report.

Set the environment variable `DCDIR` that indicates the OpenTP1 home directory for the application programs that receive a report. This value must be the same as for the OpenTP1 that reports a message log.

If the online jobs of OpenTP1 obtain all the message logs at start or later, the application programs that receive a report must be started before OpenTP1.

### **(2) Receiving a message log report**

The application programs that receive a message log report contend for the start of reception using the `dc_log_notify_open()` function. Then, they receive a message

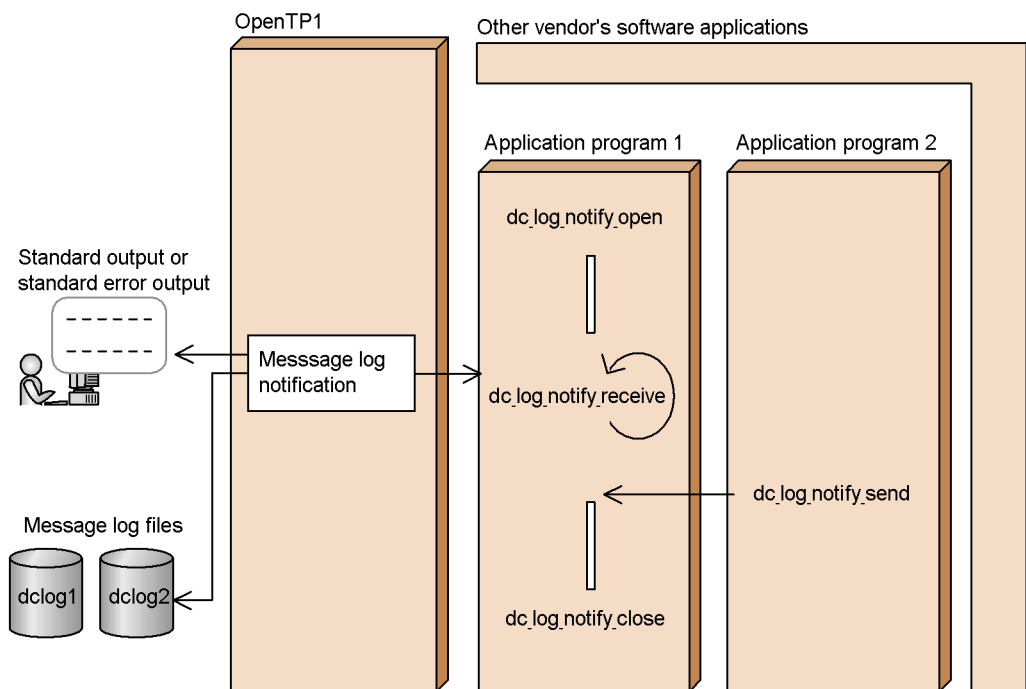
log with the `dc_log_notify_receive()` function. Only one message log can be received by the `dc_log_notify_receive()` function. To receive multiple message logs, call the `dc_log_notify_receive()` function repeatedly.

To terminate message log report reception, call the `dc_log_notify_close()` function. When calling the `dc_log_notify_open()` function after calling the `dc_log_notify_close()` function, the message log report can be re-received.

The application programs that receive a report continue to wait until the `dc_log_notify_close()` function is called after OpenTP1 has terminated. If the termination of reception is reported to the application programs, send data using the `dc_log_notify_send()` function from other application programs. In the application programs that report a reception termination, the `dc_log_notify_open()` function cannot be called before calling the `dc_log_notify_send()` function.

Figure 3-72 shows reception of a message log report.

Figure 3-72: Reception of a message log report

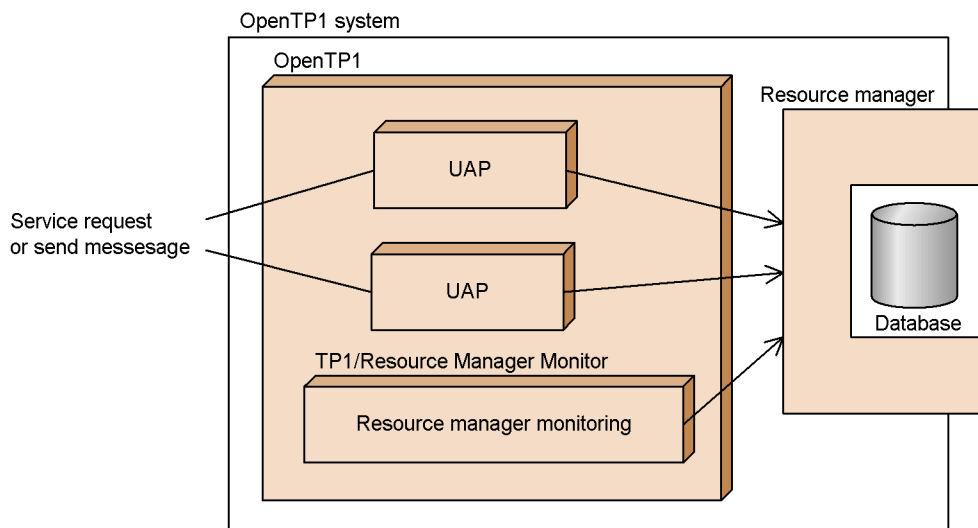


### 3.9.6 Controlling resource managers not provided by OpenTP1

You can use the TP1/Resource Manager Monitor to control the start and termination of resource managers not provided by OpenTP1.

Figure 3-73 gives an overview of resource manager control.

Figure 3-73: Overview of resource manager control



### (1) Preparing to monitor a resource manager

In the `rmm_check_services` operand of the RMM service definition, the user identifies the resource managers he or she wants to monitor. The user creates commands in the shell file to manipulate resource managers monitored by the RMM service. The user then specifies the file names of the created commands (listed below) in the monitored RM definition.

- Monitored RM start command (`rmm_start_command` operand)
- Monitored RM termination command (`rmm_stop_command` operand)
- Monitored RM forced termination command (`rmm_abort_command` operand)
- Monitored process ID acquisition command (`rmm_get_pid_command` operand)

Samples for the above commands are provided. Modify the samples according to your job so you do not need to create commands from scratch. Samples are stored in the `$DCDIR/etc/RMmonitor/` directory.

For details about creating commands, see the manual *OpenTP1 Operation*. For details about definitions, see the manual *OpenTP1 System Definition*.

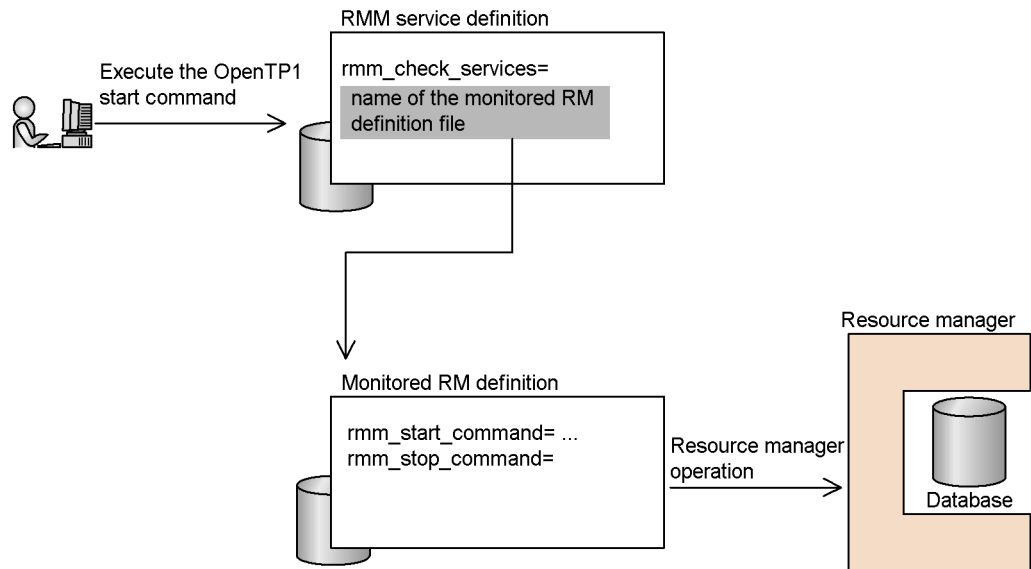
### (2) Time to start monitoring

You can create a definition for monitoring a resource manager so that resource manager operation starts automatically when OpenTP1 starts.

Figure 3-74 shows how the start of OpenTP1 is related to the RMM service definition

and the monitored RM definition.

Figure 3-74: Start of OpenTP1 and the definitions to be referenced



### 3.9.7 Uptime statistics

System statistics let you view the operation status of OpenTP1.

#### (1) *Outputting statistics*

You can use the `jnlstts` and `jnlmcst` commands to extract statistical journal (SJ) information from an unloaded-journals file, edit it to help check the OpenTP1 operation status, and output the edited data in a listing to the standard output. An OpenTP1 administrator can examine the output statistics to check the operating status of OpenTP1.

You can use `jnlstts` to output statistics about system services, UAP operation status, or transaction processing; or you can use `jnlmcst` to output MCF statistics about the sending or receiving of messages. For details about the statistics, see the manual *OpenTP1 Operation*.

The facilities provided by the `jnlstts` and `jnlmcst` commands are referred to as the *statistics output facilities*.

#### (2) *MCF system statistics*

You can collect MCF statistics into shared memory, and then output them to a UNIX file as statistical data. You can also edit and output the statistics from the UNIX file to the standard output.

Use *MCF system statistics* to check the operating status of MCF.

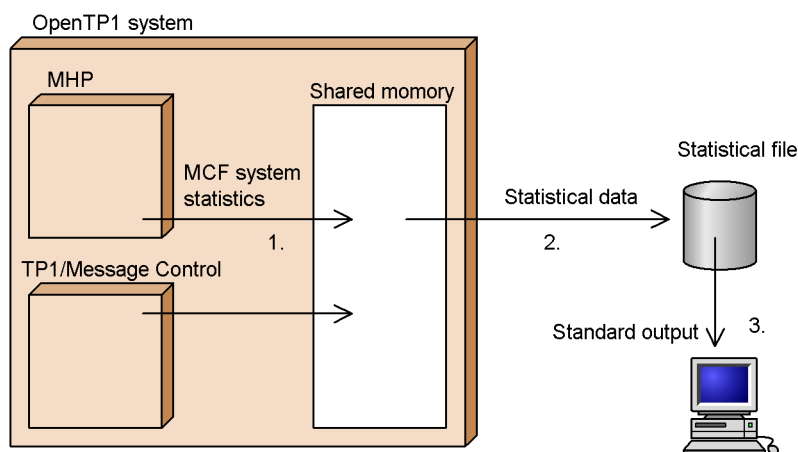
The MCF system statistics described here are independent of the `jnlmcst` command, which acquires MCF system statistics based on journal files. The command provides information such as the number of messages processed within a specified period of time. The MCF system statistics provide information such as the number of messages waiting to be processed.

The following facilities are available for handling MCF system statistics. Facility numbers correspond to those in the Figure 3-75.

1. Facility for acquiring MCF statistics (the facility enabled by specifying `yes` in the `stats` operand of the `-w` option in the `mcfmcomn` definition command in the MCF manager definition)
2. Facility for outputting MCF statistics (`mcfstats` command)
3. Facility for editing MCF statistics (`mcfreport` command)

Figure 3-75 provides an overview of the facilities for MCF system statistics.

Figure 3-75: Overview of the facilities for MCF system statistics



### (3) Outputting system statistics in real time (`dcreport` command)

The `dcreport` command lets you output system statistics to the standard output in real time, and also allows for output of system statistics to the message log. System statistics can also be output to the standard output in CSV format. This command can be used when the `statistics` operand is set to `Y` in the system common definition, or when the `dcstats` command is used to make a request for outputting system statistics to journal files.

When you acquire system statistics by using the `dcreport` command, you can view

the service execution status, the resource load status, the error status, as well as other information, in real time.

### 3.9.8 Real-time statistics service

The real-time statistics service acquires statistics for the entire system, for each server, and for each service. By outputting the acquired real-time statistics to the standard output or log files, you can check the operating status of the OpenTP1 system in real time, and can perform system operation management and error recovery more quickly.

This subsection provides an overview of the real-time statistics service. For details about operation using the real-time statistics service and the real-time statistics that can be acquired, see the manual *OpenTP1 Operation*.

The real-time statistics service uses the following UAPs:

#### RTSSUP

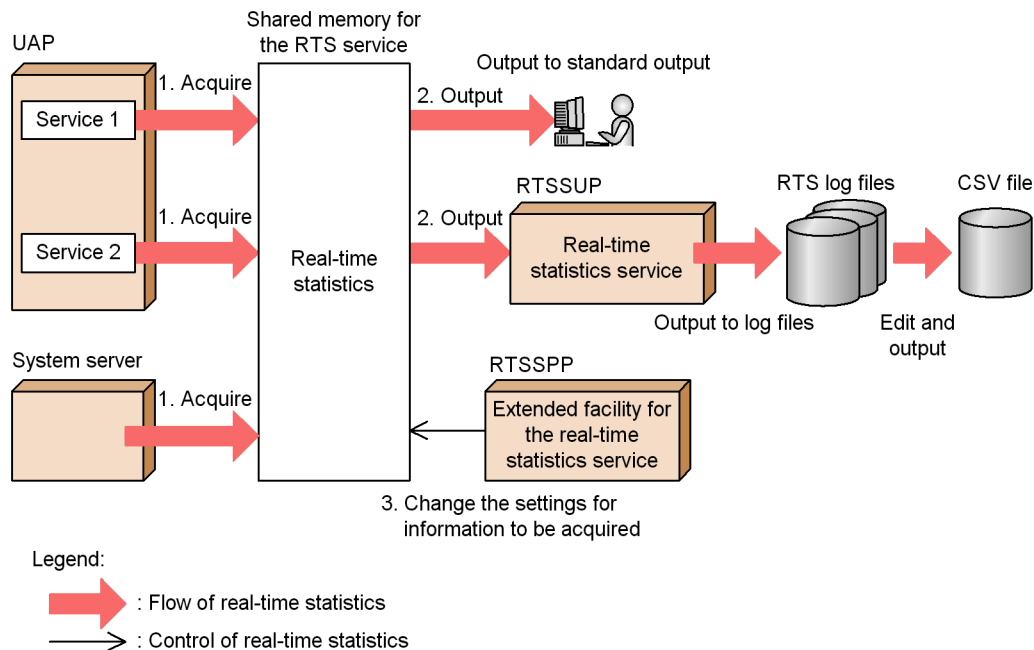
RTSSUP is the UAP to be activated when the real-time statistics service is used. RTSSUP can acquire and output real-time statistics.

#### RTSSPP

RTSSPP is the UAP to be activated when the extended facility of the real-time statistics service is used. The extended facility allows the user to change the settings for the information to be acquired while the real-time statistics service is running.

The following figure shows an overview of the real-time statistics service.

Figure 3-76: Overview of the real-time statistics service



The functionality of the real-time statistics service is described below. The paragraph numbers correspond to the numbers in the figure above.

1. The predefined real-time statistics items are acquired for the entire system, for each server, and for each service.

Acquired real-time statistics are stored in the shared memory for the RTS service. For details about the shared memory used by the real-time statistics service, see *7.2.2(5) Shared memory used by real-time statistics service*.

The execution time of a specific section in a UAP can also be acquired as real-time statistics. For details about acquiring real-time statistics for a specific section in a UAP, see the *OpenTPI Programming Guide*.

2. The real-time statistics stored in the shared memory for the RTS service are output.
  - The real-time statistics can be output to the standard output in real time.
  - The real-time statistics can be output to RTS log files.
  - The real-time statistics output to RTS log files can be edited and the editing results can be output as a CSV-format file.
3. The settings for the information to be acquired can be changed while the real-time

### 3. Functions

statistics service is running.



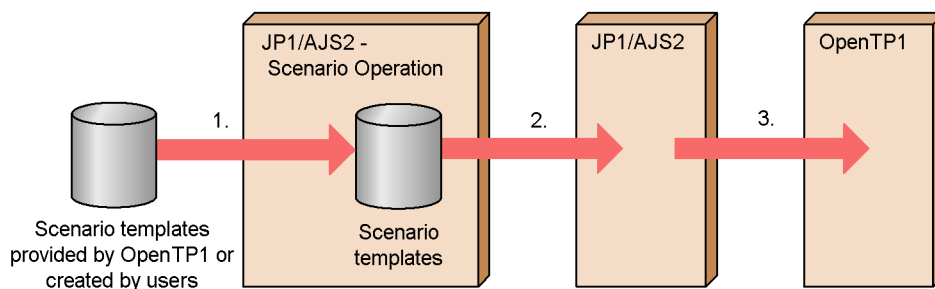
## 3.10 System operations using scenario templates

Users can prepare scenario templates that define OpenTP1 operations. Scenario templates are managed by JP1/AJS2 - Scenario Operation, and can be used to automate system operations. JP1/AJS2 - Scenario Operation executes scenario templates via JP1/AJS2 - Manager. OpenTP1 provides standard scenario templates in which basic operations are defined. These standard scenario templates make it easier for you to automate a system operation.

For details on JP1/AJS2 - Scenario Operation, see the manual *Job Management Partner 1/Automatic Job Management System 2 - Scenario Operation*.

The following figure shows the concept of using scenario templates for an operation.

Figure 3-77: Concept of using scenario templates for a system operation



Note:

Scenarios can be executed in different OpenTP1 environments by changing input scenario variables of the scenario templates.

Explanation:

1. Register scenario templates provided by OpenTP1 or created by users in JP1/AJS2 - Scenario Operation.
2. JP1/AJS2 - Scenario Operation registers a scenario in JP1/AJS2.
3. JP1/AJS2 lets OpenTP1 execute the scenario.

OpenTP1 provides scenario templates for the following scenarios:

- Scale Out

This scenario creates a new OpenTP1 node, and adds the node to the domain configuration of the OpenTP1 system.

- Scale In

### 3. Functions

This scenario releases resources of the least heavily loaded nodes on a task or node basis, and allocates the resources to other systems.

- Rolling Update

This scenario applies security-enhancing patches to the OS or UAPs without interrupting the system operation.

For details on scenario templates, see the manual *OpenTPI Operation*.

### 3.11 System monitoring using audit logs

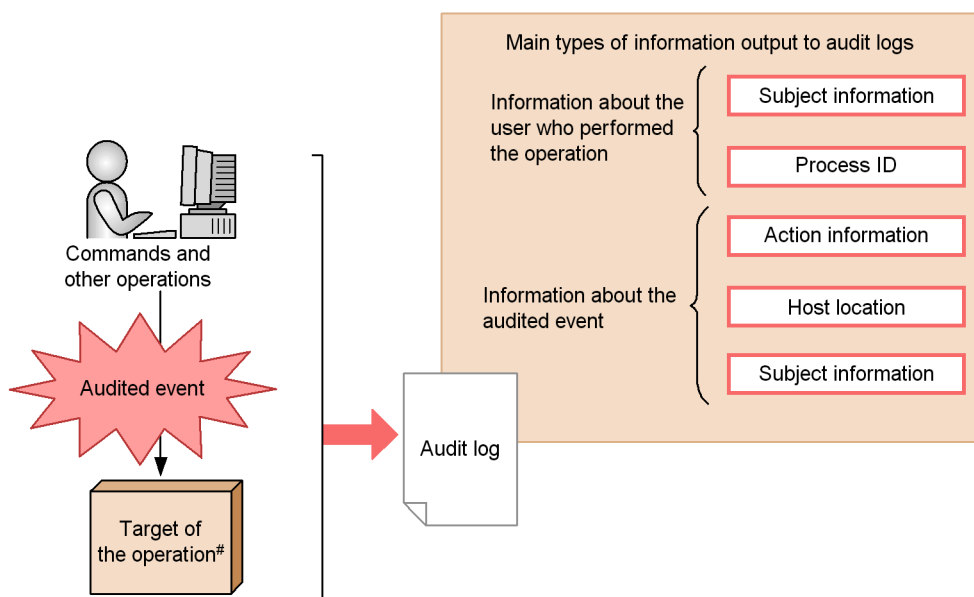
An *audit log* is a file containing historical information about the operations performed on OpenTP1 programs by system developers, operators, and users, together with the program behavior triggered by those operations. By examining an audit log, the auditor can find out what was done, when, and by whom. Thus, audit logs can be used as records for investigating system usage and unauthorized access.

The entries in an audit log include information about the user who executed a command or performed an operation, information about audited events such as whether the processing resulting from an operation succeeded or failed, and information about the object of an operation or process. This information is useful for monitoring the system.

When OpenTP1 is linked with JP1/NETM/Audit, audit logs can be automatically collected and centrally managed.

Figure 3-78 shows the flow of audit logging and the main types of information acquired.

Figure 3-78: Audit logging and main categories of information acquired



#: Target of a command or other operation, such as the server that is started when a server start command is executed

Information is output to an audit log when an operation is performed in an OpenTP1 program, such as execution of a command. Logged operations may be executed by a system administrator or operator, according to the task being performed. An entry is also logged when an audited event occurs in a process. *Audited events* are OpenTP1 program operations and resultant processing that need to be recorded to examine the adequacy of the system configuration, operation, and usage. Audited events are categorized and defined in OpenTP1 as shown in Table 3-21.

Table 3-21: Definition of audited events

Event category	Event description	Output information	
StartStop	Event indicating that software was started or stopped <ul style="list-style-type: none"> <li>• OpenTP1 started or stopped</li> <li>• User server startup or termination</li> </ul>	Start	Software started
		Stop	Software stopped
Authentication	Event indicating whether attempted authentication by a client user succeeded or failed	Login	User logged in
		Logout	User logged out
		Logon	User logged on
		Logoff	User logged off
		Disable	Account disabled
AccessControl	Event indicating whether attempted access by an administrator or user to a controlled resource succeeded or failed	Enforce	Access controlled
ConfigurationAccess	Event indicating whether an operation by an administrator or user to change or otherwise manipulate setting information succeeded or failed	Refer	Information referenced
		Add	Information added
		Update	Information updated

Event category	Event description	Output information	
		Delete	Information deleted
Failure	Event indicating a software error	Occur	Error occurred
LinkStatus	Event indicating whether equipment is linked	Up	Link active
		Down	Link inactive
ExternalService	Event indicating the result of communication between the software and an external service	Request	Request
		Response	Response
		Send	Send
		Receive	Receive
ContentAccess	Event indicating whether attempted access to critical data succeeded or failed	Refer	Information referenced
		Add	Information added
		Update	Information updated
		Delete	Information deleted
Maintenance	Event indicating whether a maintenance operation by an administrator or technician succeeded or failed	Install	Software installed
		Uninstall	Software uninstalled
		Update	Software updated
		Backup	Data backup
		Maintain	Maintenance task
AnomalyEvent	Event indicating a communication error	Occur	Error occurred

### 3. Functions

Event category	Event description	Output information	
ManagementAction	Event indicating a critical action in a program, or an action triggered by a different category of audited event	Invoke	Administrator called a function
		Notify	Administrator was notified

Events are defined for each audit event category. For a detailed list of audit events, see the description of logged event information in the manual *OpenTP1 Operation*.

OpenTP1 provides an API (`dc_log_audit_print` function) that outputs specified audit log data from a UAP. Using this API, you can output audit log entries when a UAP operation is performed or processing is performed by the UAP, as well as when an audit event occurs.

For details about logging selected audit information, see the *OpenTP1 Programming Guide*.

## Chapter

---

# 4. File System

---

This chapter provides an overview of the OpenTP1 file system and describes OpenTP1 files (system files, queue files, and user data files) and the IST service.

- 4.1 The OpenTP1 file system
- 4.2 System files
- 4.3 Queue files
- 4.4 User data files

---

## 4.1 The OpenTP1 file system

---

The OpenTP1 uses both the file system provided by the OS (operating system) and the file system provided by OpenTP1. The OpenTP1 file system provides high reliability with extensive features for duplicating information in important files.

### 4.1.1 Overview of the OpenTP1 file system

The *OpenTP1 file system* is specific to OpenTP1 and is distinct from the OS file system. An OpenTP1 administrator must decide where to create the OpenTP1 file system. The file system can be created on ordinary-file directories in the operating system or on character special files. If the OpenTP1 administrator wishes to separate the OpenTP1 file system from the OS system, he or she can install the two file systems in separate disk partitions or on different disks.

#### (1) The OpenTP1 file system and OpenTP1 files

The *OpenTP1 file system* is specific to OpenTP1 and is distinct from the OS file system. It can consist of character special files or regular files. User data files, journals needed for OpenTP1 system recovery, and files of critical information related to system reliability can be created in the OpenTP1 file system. They are referred to as *OpenTP1 files*.

OpenTP1 files are listed in Table 4-1.

Table 4-1: List of OpenTP1 files

File type	Purpose
Status files	Store information about system service activities and the system configuration. This information is used for OpenTP1 recovery when a failure occurs.
System journal files	Store history information about transaction processing. This information is used for OpenTP1 recovery when a failure occurs. Also store information about UAP processing.
Checkpoint dump file	Stores status-table information needed for recovery. This information is used for OpenTP1 recovery when a failure occurs.
Archive journal file	Stores collected node journals in a cluster system or parallel processing system configured using TP1/Multi.
Message queue file	Stores messages waiting to be sent or received using the message exchange facility. TP1/Message Control must be installed in the system.
MQA message queue file	Stores a waiting queue for messages used by MQA message queuing. TP1/Message Queue must be installed in the system.



File type	Purpose
DAM files	Used as user files. TP1/FS/Direct Access must be installed in the system.
TAM files	Used as user files. TP1/FS/Table Access must be installed in the system.

Note

The following files can also be created to assist in OpenTP1 file management:

- Transaction recovery journal file  
Used to reduce the size of journals for user-specified transactions.
- Server recovery journal file  
Stores information about services. This information reduces recovery time at a complete-recovery restart.

Figure 4-1 shows the relationship between the OpenTP1 file system and the OS file system. Table 4-2 describes the differences between the two file systems. Figure 4-2 shows how to select files to create an OpenTP1 file system.

Figure 4-1: Possible locations of an OpenTP1 file system

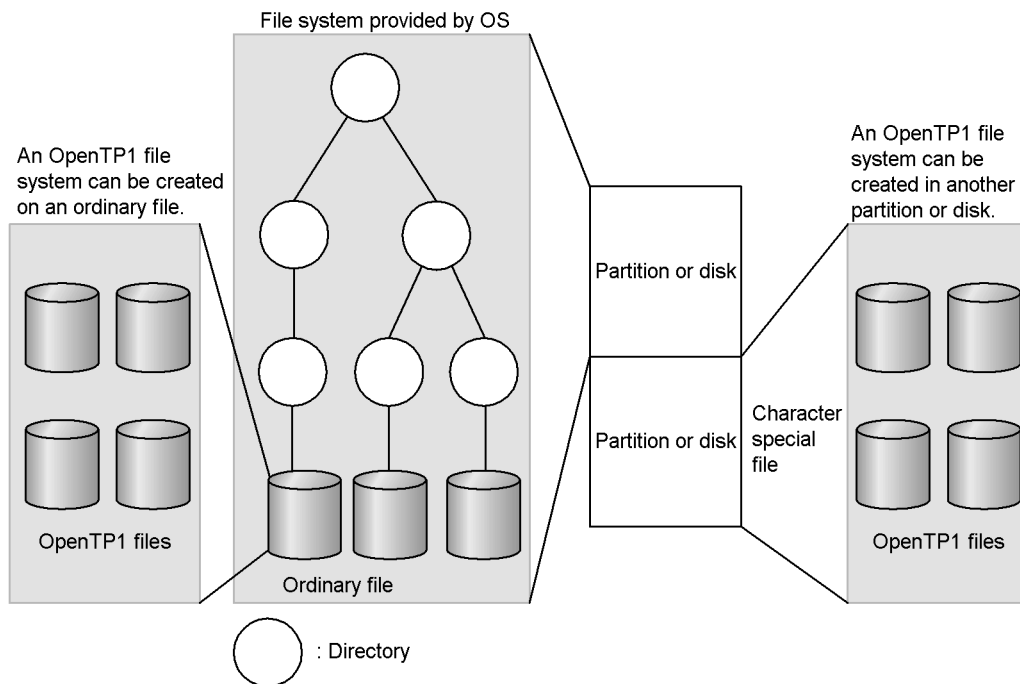
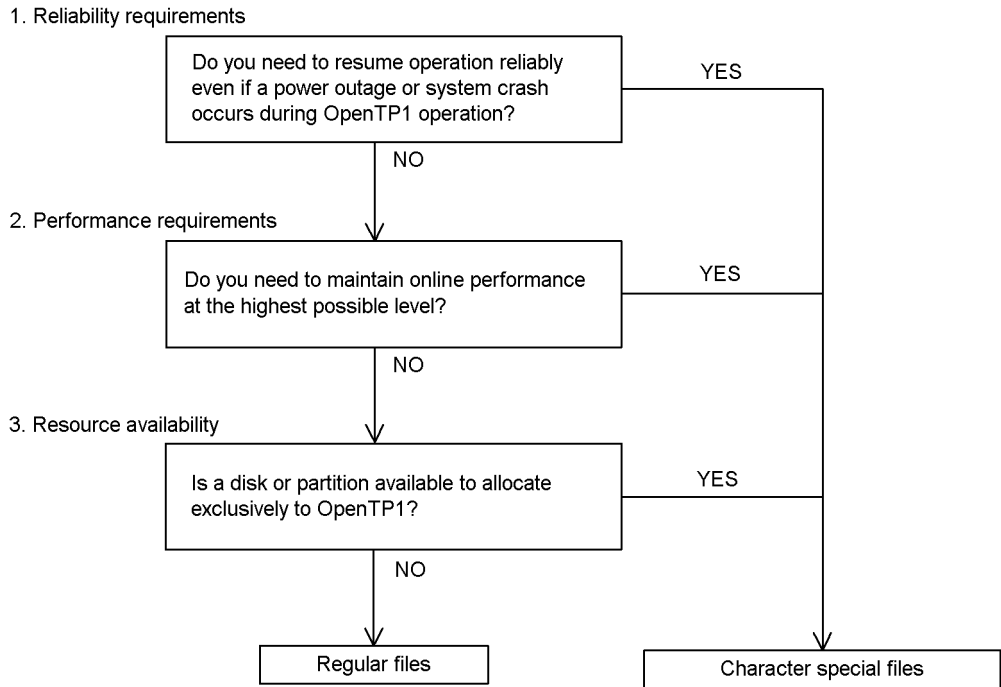


Table 4-2: Differences between the OpenTP1 file system and the OS file system

Comparison point	OpenTP1 file system		OS file system
	Character special file	Ordinary file	
(Reliability) What happens to data specified by a write request that normally terminates immediately before OpenTP1 abnormally terminates?	The data is written.	← Same.	Undefined.
(Reliability) What happens to data specified by a write request that normally terminates immediately before a process abnormally terminates?	The data is written.	← Same.	Undefined.
(Reliability) Security of disk files in the event of system shutdown for a reason such as a power outage	Security is high in terms of maintaining file integrity because management information is not stored in a separate area on the disk.	Security is lower than for character special file because management information is stored in a separate area on the disk.	Security is low in terms of file integrity because management information is stored in a separate area on the disk.
(Reliability) How is a file area allocated?	A shortage of space never occurs during online operation because a file area is allocated beforehand when the file system is created.		A shortage of space might occur in an extension of write operation because a block is allocated when a file is added.
(Performance) How efficient is usage of disk area?	If an area has been allocated to a file, it cannot be allocated to another file even if there are no OpenTP1 file records written in the area.		Because the disk area is allocated dynamically block by block as the file size increases, the efficiency of disk area usage is higher than for an OpenTP1 file system.
(Performance) How fast is read time (time until the function is returned)?	There is no buffer cache advantage because buffer cache is not used.	Read time is generally fast because if the target data is found in the buffer cache, the system does not have to search the disk. However, if a process uses the buffer for control at a higher level, overhead might increase due to double buffering.	Read time is generally fast because if the target data is found in the buffer cache, the system does not have to search the disk.

Comparison point	OpenTP1 file system		OS file system
	Character special file	Ordinary file	
(Performance) How fast is write time: the time until the function is returned?	Write time is generally slower than for an OS file system because I/O operations are performed synchronously, but the overhead for write processing is low because the buffer cache is not used.	Write time is generally slower than for an OS file system because I/O operations are performed synchronously.	Write time is generally fast because I/O operations are performed asynchronously.
(Performance) How even is access time?	Access time is close to even because a contiguous area is allocated.	Access time is uneven because areas are not always contiguous.	Access time is uneven.
(Operations) Does file size need to be estimated?	Size must be specified during file creation.		Size does not need to be specified when creating the file.
(Operations) Is a dynamic secondary allocation of file size possible?	Not possible.		Possible.
(Operations) Can files be sorted according to directory?	Not possible.		Possible.

Figure 4-2: Selection of files to create the OpenTP1 file system



**(2) Regular files**

The advantage of regular files is that they are flexible and efficient to use. This is why they are used for OpenTP1 definition files and so on.

Table 4-3 describes the regular files used in OpenTP1.

Table 4-3: Regular files used in OpenTP1

File type	Purpose	Remarks
User program file	Stores a UAP executable program.	Created by the user.
Definition file	Stores OpenTP1 system definitions. A definition file can be created as a text file, using a text editor provided by the OS.	
Map file	Used by the mapping service. Stores physical maps and pre-loaded maps.	

File type	Purpose	Remarks
OpenTP1 program file	Store an OpenTP1 program, including the executable file and files used to create the UAP.	Created automatically when the program is installed.
Definition analysis file	Used internally in OpenTP1 to analyze definitions.	
Message object file	Stores a message text.	
Command log file	Stores an OpenTP1 command log.	
Message log file	Stores system messages output by OpenTP1.	Created at OpenTP1 execution.
MCF trace file	Stores MCF trace information.	
Schedule queue information file	Stores schedule queue information internally in OpenTP1.	
RPC trace file	Stores an RPC trace.	
Trace information dump file	Stores internal OpenTP1 trace information.	
Shared memory dump file	Stores a shared memory dump output by OpenTP1.	
Core file	Stores the core of an abnormally ended process.	
Deadlock and timeout information file	Stores deadlock and timeout information.	
MCF dump file	Stores a MCF dump.	
MCF shared memory dump file	Stores dump information, output when an error occurs, about the shared memory area allocated to the MCF.	
Undetermined-transaction information file	Stores information about an undetermined transaction, output when an error occurs.	
Invalid journal information file	Stores invalid journal information detected when a journal is read.	
Copy file of input/output queue contents	Stores the queue contents when the command to copy the contents of the input/output queue is executed.	
Trace information collection file for verification	Stores trace information for performance verification.	
XAR performance verification trace information file	Stores trace information for events generated during transactional linkage using the XA resource service.	

File type	Purpose	Remarks
JNL performance verification trace information file	Stores trace information for the journal service.	
LCK performance verification trace information file	Stores trace information for events generated during locking using the lock service.	
MCF performance verification trace information file	Stores trace information for events generated during message transmission using MCF.	
TRN event trace information file	Stores trace information for events generated by the transaction service and the XA function called at a transaction branch.	
NAM event trace information file	Stores trace information for events relating to the name service, including communication invoked by the name service and the registration and deletion of service information from the cache.	
Process service event trace information file	Stores trace information for a process service.	
FIL event trace information file	Stores event information if the time required to process an OpenTP1 file access request is equal to or greater than the value specified in the <code>fil_prf_trace_delay_time</code> operand in the system common definition.	
RTS log file	Stores real-time statistics.	
UAP trace edit/output file	Stores a UAP trace that is edited and output automatically when a UAP terminates abnormally.	
OpenTP1 debugging information file	Stores OpenTP1 information when a UAP terminates abnormally.	

### 4.1.2 Creating an OpenTP1 file system

In general, to create the OpenTP1 file system, an OpenTP1 administrator:

1. Uses the `filmkfs` command to create the OpenTP1 file system on a character special file or ordinary files.
2. Uses the following commands to allocate physical files for each file type:
  - Status file: `stsinit` command
  - System journal file and checkpoint dump file: `jnlinit` command
  - MCF message queue file: `queinit` command
  - MQA message queue file: `mqainit` command

DAM file: `damload` command

TAM file: `tamcre` command

3. Uses system service definitions to make the physical files, which are the actual files used for input or output, usable as an OpenTP1 file. For example, in the status service definition you can define which physical files make up the logical filegroup that is the OpenTP1 status file.

The following figure shows an example of the complete pathname of a physical file created in the OpenTP1 file system on a character special file.

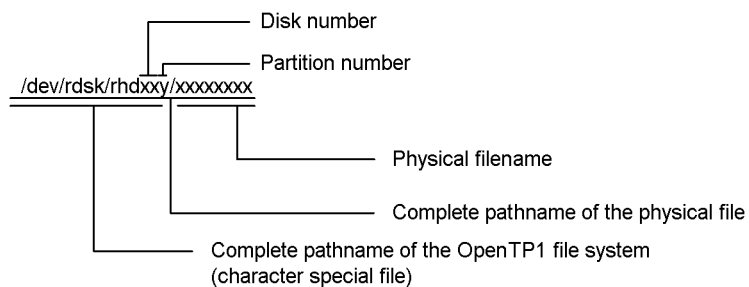
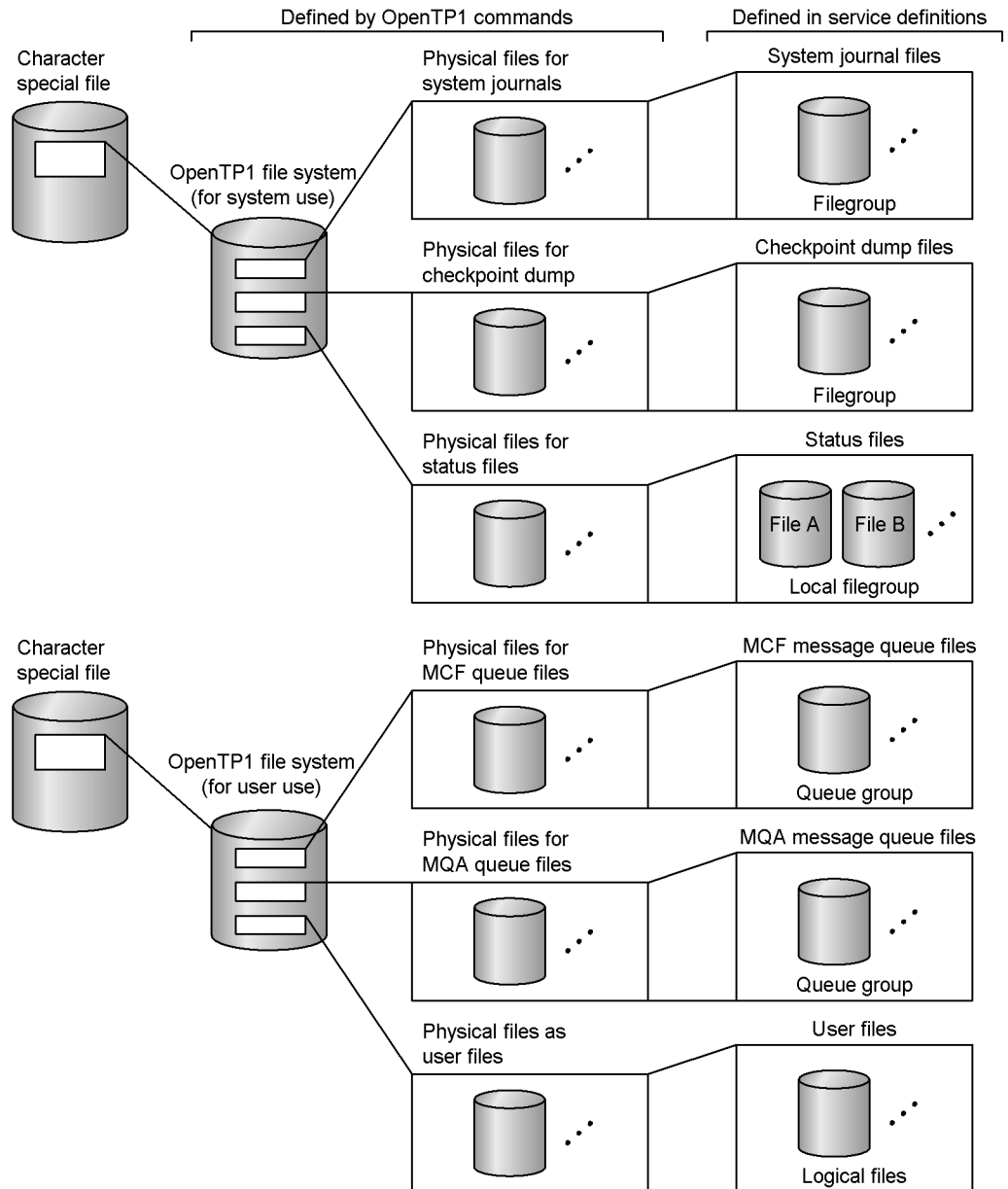


Figure 4-3 shows an example of an OpenTP1 file system. In this example the OpenTP1 file system has separated the OpenTP1 system files from the user files. Separating the OpenTP1 file system into subsystems for system files and user files enhances system reliability and performance.

Figure 4-3: Separating the OpenTP1 file system into OpenTP1 files and user files on character special files



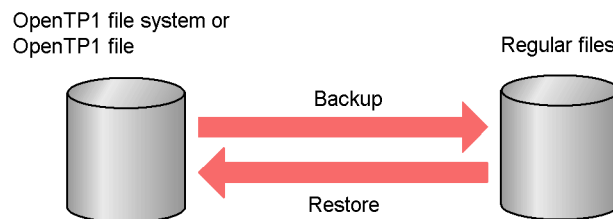


### 4.1.3 Backing up and restoring OpenTP1 file systems

OpenTP1 files contain important information that is used in situations such as recovery of an OpenTP1 process. To prevent OpenTP1 files from being irrecoverably corrupted, the user should periodically backup each OpenTP1 file system or each OpenTP1 file. For example, the backups can be made whenever OpenTP1 starts. After a backup has been made, even if OpenTP1 files are destroyed or corrupted, the user can use the backups to restore the files.

The following figure illustrates the process of backing up and restoring an OpenTP1 file system.

Figure 4-4: OpenTP1 file system backup and restore



User files contain data important for applications. OpenTP1 provides the DAM FRC facility and related commands to guarantee such individual items of data. Using this facility and related commands you can:

- Back up or restore user-dedicated files.
- Restore a user file, based on a backup or based on a file copied from a system journal file.

For details about backing up and restoring or about the procedure for restoring user files, see the manual *OpenTP1 Operation*.

### 4.1.4 Protecting OpenTP1 files

User's mistakes must be prevented from corrupting files. In particular, the OpenTP1 system administrator must protect files containing important data: for example, data such as control information required to continue online processing or information critical to company functioning.

#### (1) Protection for each OpenTP1 file system

An OpenTP1 administrator can use OS commands to manage the file owners and file access authorizations for each file system, which helps prevent unintentional corruption of files. An OpenTP1 administrator can:

- classify file owners as the OpenTP1 administrator or an OpenTP1 group

- provide users with restricted file access authorization: either READ-WRITE or READ

These features are made more effective and the OpenTP1 file system is easier to manage if the OpenTP1 administrator separates the OpenTP1 file system into two subsystems:

- a subsystem in which the files contain OpenTP1 system-related items
- a subsystem in which the files are user files

For safety and convenience, access to system files can be restricted to the OpenTP1 administrator and the users of the OpenTP1 group. This subsystem should contain files used by OpenTP1 to provide uninterrupted online processing: for example, the system journal file, checkpoint dump file, and status file.

For convenience to end-users, user files should be created in another subsystem of the OpenTP1 file system.

Table 4-4 shows an example of using owner and access authority to protect files when the OpenTP1 file system separates system and user files.

*Table 4-4:* Example of OpenTP1 file system protection (by owner and access authority)

OpenTP1 file system	Owner		Access authority		
	User ID	Group ID	Owner	User in the group	Other user
For system	OpenTP1 administrator	OpenTP1 group	rw (permitted to read and write)	r- (permitted to read)	r- (permitted to read)
For user	OpenTP1 administrator	OpenTP1 group	rw (permitted to read and write)	rw (permitted to read and write)	r- (permitted to read)

## **(2) Protection for individual OpenTP1 files**

An OpenTP1 administrator can use the commands `filchown` and `filchmod` during online processing to dynamically change the owner of, and access authority for, an OpenTP1 file. This enables specific users to be validated to use an OpenTP1 file during online processing. For details about protecting OpenTP1 files, see the manual *OpenTP1 Operation*.

### **4.1.5 Assigning an OpenTP1 file system**

You can improve efficiency and reliability by creating OpenTP1 file systems with a disk assigned to each type of file.

If you create each type of file on a different disk, the cost (for the disks) increases, but

efficiency and reliability also increase. If you create the entire file system on one disk, the cost decreases, but reliability also decreases because if the disk become unusable, all the files become unusable.

To create the most appropriate file system, the OpenTP1 administrator needs to decide on the basis of a variety of factors including cost, operability, efficiency, and reliability.

The following are typical factors to consider when deciding how to organize files:

- **Hardware configuration**

When creating an OpenTP1 file on a character special file and when the disk configuration is determined, the OpenTP1 administrator must know the number and sizes of the partitions that can be allocated for an OpenTP1 file system.

- **System configuration**

The OpenTP1 administrator must:

- Determine what types of files are necessary by considering the functions of the system to be configured. For example, the message queue file is unnecessary if messages need not be exchanged with the host computer.
- Determine the sizes of files such as the system journal file and checkpoint dump file by considering such factors as how many transactions are processed in one day. Also, determine the size of the message queue file by considering the number of exchanged messages and the length of each message.

- **Reliability**

More than one subsystem of the OpenTP1 file system should be created according to the importance of the files. For example, you can create one subsystem for use by the OpenTP1 system and another for use by users. The file subsystem for use by OpenTP1 could include the system journal file, checkpoint dump file, and status file. The file subsystem for use by users could include the message queue file and user files.

- **Performance**

If frequently accessed files are all on the same disk, data I/O performance will deteriorate. To prevent this, place frequently accessed files on different disks.

The OpenTP1 administrator needs to consider the above factors, and decide the most appropriate file system on the basis of performance, reliability, cost, and operability considered as a whole.

---

## 4.2 System files

---

OpenTP1 collects internal information in the following types of system files in case an error occurs:

- Status files
- System journal files
- Checkpoint dump files
- Transaction recovery journal file(TRF)
- Server recovery journal file(SRF)
- Archive journal files

### 4.2.1 System files: status files

#### (1) Purpose of status files

A *status file* stores information about OpenTP1 service activities and the OpenTP1 system configuration. This information is used for recovering OpenTP1 if a failure occurs. A status file stores data about system operations (such as what commands were entered and what effects the commands produced); so examining status files can help the OpenTP1 administrator or programmer to find out the cause of some abnormal termination. Status files also store the data necessary for automatically restarting OpenTP1.

The data that must be retained to restart OpenTP1 is called *system control information*. A status file stores system control information each time the status changes. Some items of system control information are:

- termination status of OpenTP1
- operation status for OpenTP1 services or UAPs: such as whether a service or UAP is starting, executing, or terminating
- whether each system file and user file is open or closed
- connection status for devices connected with MCF

#### (2) Structure of status files

The information in a status file is critical for restarting OpenTP1 so OpenTP1 duplicates this information within the status file. A status file is thus actually a *logical filegroup*, consisting of a pair of files: *physical file A* and *physical file B*. Even if an error occurs in one of the physical files, OpenTP1 can still be restarted by using the information in the other physical file.

To avoid both physical files being corrupted at the same time, place physical file A and

physical file B on different disks if possible. Note that for the same status file, physical file A and physical file B must have the same size and record length.

**(a) Operating with only one physical file in a status file**

Usually, if a failure occurs in one physical file of a status file during online operation and no standby logical filegroup is found, OpenTP1 terminates abnormally. In the status service definition, however, you can specify whether to permit OpenTP1 to continue operation using the one normal physical file (either physical file A or B) only. In some manual versions, this *operating with only one physical file in a status file* is called *one-system operation of a status file*.

There is a serious drawback to operating with only one physical file in a status file: if another failure occurs, the restart information in the single physical file is lost, making a restart impossible. Thus when an OpenTP1 message in the message log reports that the system is operating with only one physical file in a status file, the OpenTP1 administrator should quickly prepare a usable standby logical filegroup and resume normal operation.

**(3) Status of status files**

The status of an OpenTP1 status file can be:

- Current

The current status file is a logical filegroup that is open and currently assigned to store system control information. A file entity is required.

- Standby

A standby status file is a logical filegroup that is open but is not currently assigned to store system control information; the file is standing by as a substitute in case an I/O error occurs in the current status file. A file entity is required.

- Invalid

A reserved status file is a logical filegroup that is closed but defined in the status service definition. A file that has been deleted and has no entity is called an *invalid file without a file entity*. OpenTP1 cannot start if there is an invalid file without a file entity.

- Shutdown

A shutdown status file is a logical filegroup that is closed and cannot be opened because an error occurred during online processing.

When OpenTP1 starts, it opens all the files defined in a status service definition as being part of a logical filegroup: i.e., part of a status file. Of the opened status files, the files assigned to store the status information are the current files and the others are standby files.

Status information is always stored in the same status file as long as no errors occur in

it. In the current status file, OpenTP1 first writes system control information to physical file A and then to B. Therefore, even if an abnormal termination of OpenTP1 causes corruption of A files during a new write, OpenTP1 can be restarted in its former status by reading the uncorrupted B file in a complete recovery.

#### (4) Swapping status files

When status files are swapped, a standby file replaces the current status file. If an I/O error occurs in either physical file A or B, OpenTP1 copies the system control information from the undamaged file (either A or B) to the standby logical filegroup. In the standby logical filegroup, the system control information is first copied to physical file A and then to B. After the information is copied, the standby logical filegroup becomes the current logical filegroup.

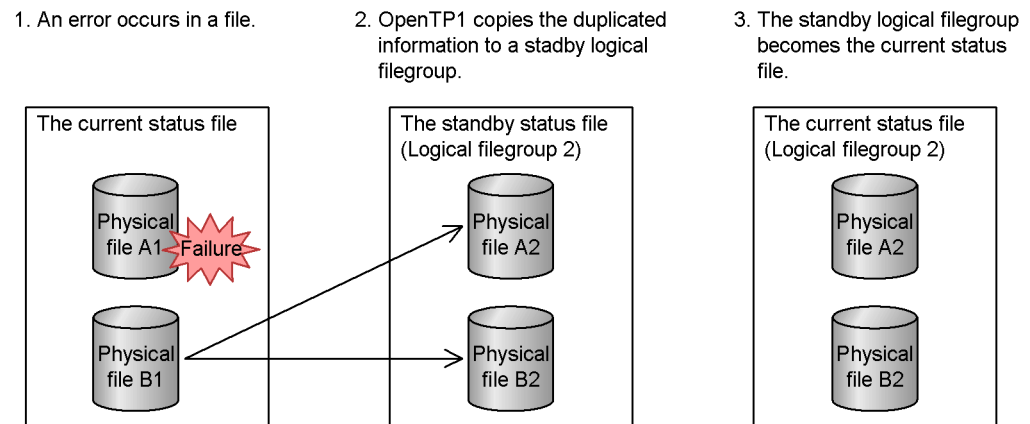
If there is no standby logical file, OpenTP1 terminates abnormally. For the processing in this case, see 5.3 *Failure and error recovery*. However, if operation with only one physical file in a status file is specified, OpenTP1 continues operation using the single operable status file.

You can use commands to delete, resize, and re-create a status file that has been shut down because of a failure. The command used for deletion (`stsrsm`), however, can delete only files that have an invalid or shutdown status; you cannot use it to delete the current or standby files.

The system control information in a status file is re-edited from the beginning of files A and B when status files are swapped. OpenTP1 automatically swaps status files as required to remove data fragmentation that might be caused by repeatedly starting and terminating OpenTP1.

Figure 4-5 illustrates how status files are swapped. For the processing if an error occurs in a status file, see 5.3 *Failure and error recovery*.

Figure 4-5: Swapping of status files



## 4.2.2 System files: system journal files

### (1) Purpose of system journal files

System journal files are used primarily for:

- system recovery (recovery journals and synchronization point journals)
- system tuning (statistical journals)
- providing information about UAPs (user journals)

The recovery and synchronization point journals store system-history information necessary either for a complete recovery when the online system stops, or for a partial recovery when a UAP transaction terminates because of an error. A complete recovery requires both of the following:

- checkpoint dump of the latest generation (described in 4.2.3 *System files: checkpoint dump files*)
- recovery journals that were obtained after collecting the checkpoint dump of the latest generation

Partial recovery of a UAP transaction requires only the recovery journals and does not require the checkpoint dump.

In addition to the journals used for recovery, the system collects various system statistics (the statistical journals), which can be used for checking and tuning the online system; and user journals, which consist of information concerning a specific UAP.

### (2) Structure of system journal files

As with the status files, the information in a system journal can be duplicated within a system journal. A system journal file is thus a *logical filegroup*, consisting of a pair of physical files: *physical file A* and *physical file B*. In such a case, OpenTP1 obtains the same journal information for physical file A and physical file B. This increases reliability because when a journal must be read because of some abnormality, even if one of the physical files is corrupted, the journal can be read from the other physical file.

To avoid the possibility that both physical files in a system journal file could be corrupted at the same time, place physical file A and physical file B on different disks if possible. It is preferable, but not necessary, for physical file A and physical file B to be the same size. It is preferable because a larger physical file cannot store a journal that exceeds the size of the smaller file.

In the system journal service definition, you must specify at least two logical filegroups (Figure 4-6). When defining a logical filegroup, you also define which physical files make up the logical filegroup. You can assign any name to a logical filegroup. Specifying a name for the logical filegroup enables you to manipulate both physical files in the logical file as a group: for example, both physical file A and

physical file B can be opened or closed at the same time.

**(3) Operating with only one physical file in a system journal file**

Usually, if a failure occurs in one physical file of a system journal file during online operation and no standby logical filegroup is found, OpenTP1 terminates abnormally. In the system journal service definition, however, you can specify whether to permit OpenTP1 to continue operation using the one normal physical file (either physical file A or B) only. When *operating with only one physical file in a system journal file* is permitted, a logical filegroup can be used as long as at least one of the two physical files that make up the filegroup is active. When operating with only one physical file is prohibited, a logical filegroup can be used only when both physical files are active.

For example, suppose there are two filegroups fg1 and fg2, consisting of the physical files fg1a, fg1b, fg2a, and fg2b. An error in a physical file is often caused by a failure in the disk device itself, and often all the physical files on the disk are damaged as a result. When operation with only one physical file is prohibited, if both physical files fg1a and fg2a in system A are damaged, there is no available filegroup and OpenTP1 terminates abnormally. After the problem is remedied, a complete system recovery will be required. When operation with only one physical file is permitted, as long as physical files fg1b and fg2b in system B are still operable, the filegroup will be available and OpenTP1 can continue operation.

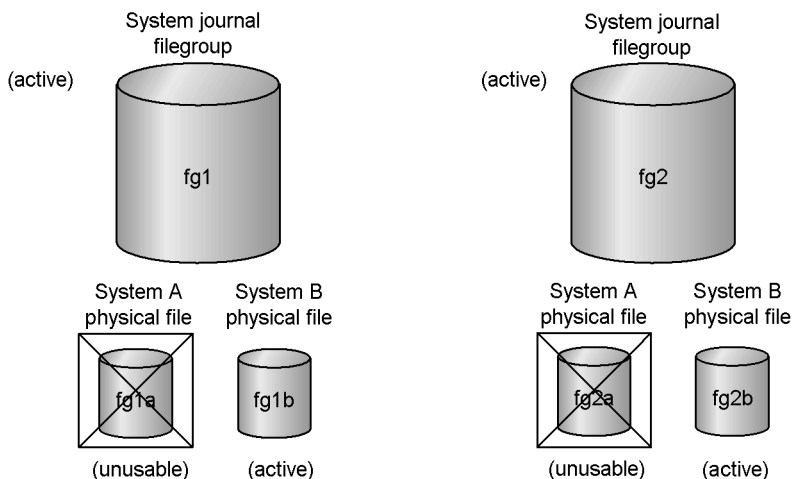
As this example illustrates, when operation with only one physical file is permitted, if one system fails, operation can continue on the other system until the problem is remedied. Note, however, that the temporary loss of redundancy between system journal files impacts on the system's reliability. Specify in the system journal service definition whether to permit or prohibit operation with only one physical file. Whether operation with only one physical file is permitted is specified in the system journal service definition.

The following figure shows the difference between the two options.

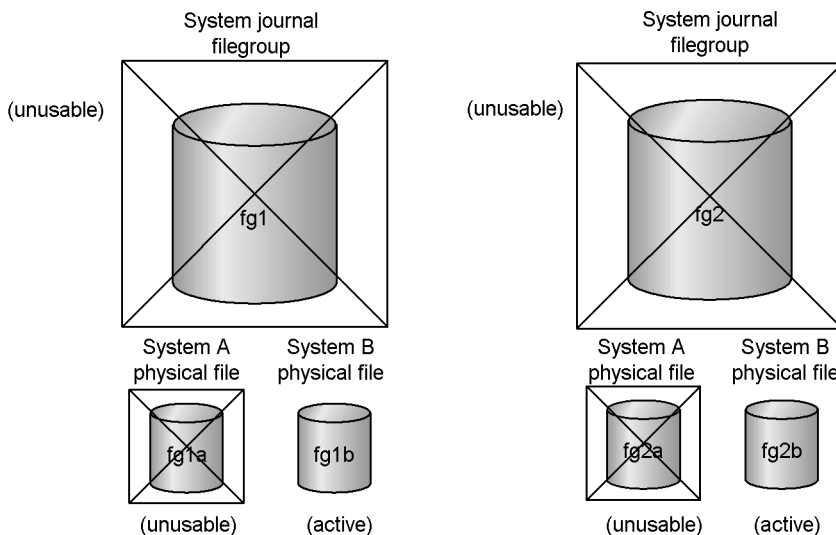


Figure 4-6: Permitting and prohibiting operation with only one physical file

● Operation with only one physical file permitted



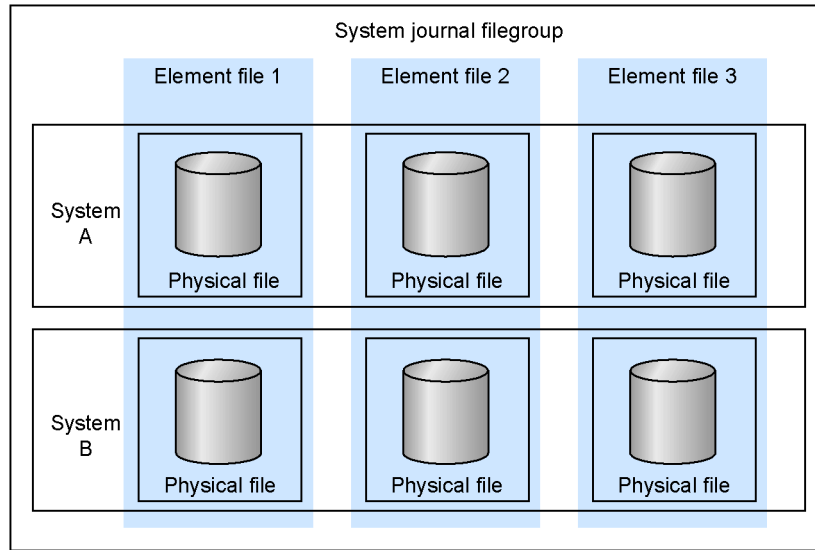
● Operation with only one physical file prohibited



**(4) Parallel access facility for system journal files**

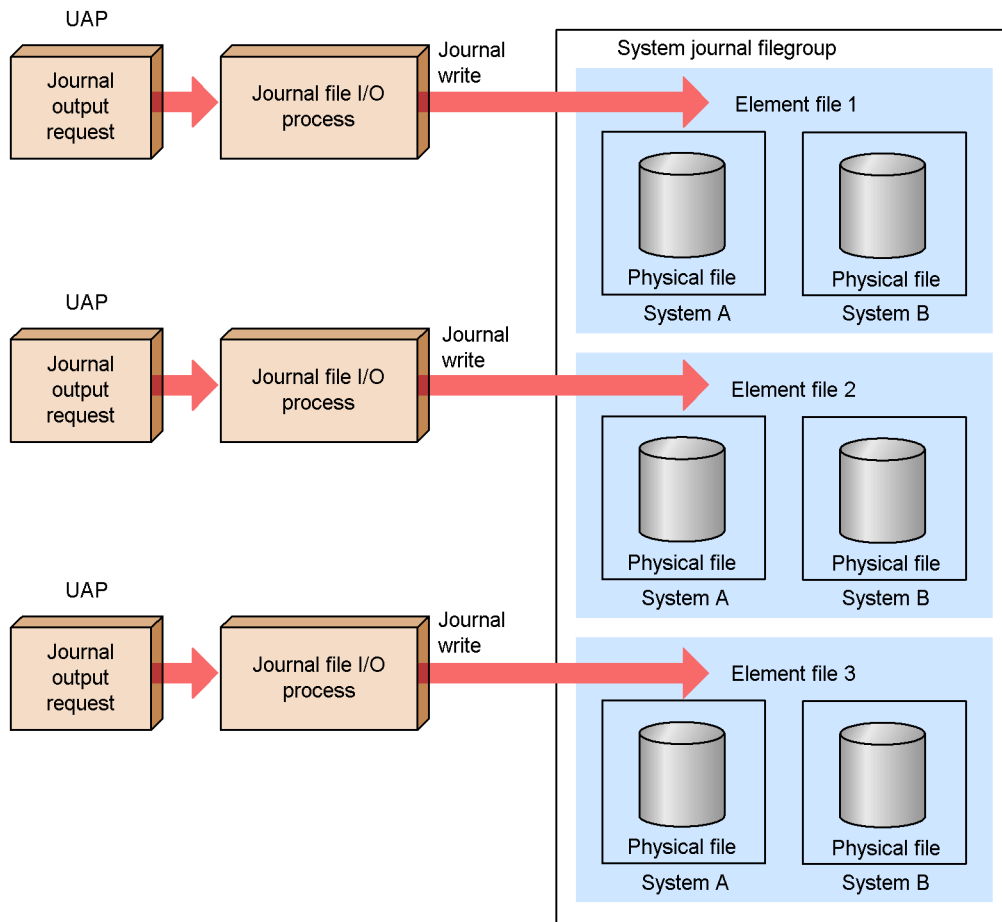
A system journal file can be configured as a filegroup containing two or more element files. This facility is known as the *parallel access facility for system journal files*. The following figure illustrates two filegroups configured redundantly using the parallel access facility.

Figure 4-7: Filegroup configuration using the parallel access facility (jnl\_max\_file\_dispersion=3)



The following figure provides an overview of the parallel access facility.

Figure 4-8: Overview of the parallel access facility  
(jnl\_max\_file\_dispersion=3)



With constant input and output to a system journal file, the disk load increases and I/O performance can deteriorate. In this situation, configuring a number of element files in a filegroup allows the system journal file to be accessed in parallel, raising the journal I/O performance.

The physical files that make up a filegroup can use the same SCSI interface and hard disk. However, this means that the files cannot be accessed in parallel, and the parallel access facility cannot be used to best effect. Therefore, a different SCSI interface and hard disk should be used for each physical file. The files do not need to be the same size, but if they are different, OpenTP1 regards the size of the smallest physical file as the size of all the element files. To use resources effectively, we recommend that you use physical files of the same size if at all possible.

When using the parallel access facility for system journal files, the number of element files that can be accessed in parallel (hereafter referred to as the *number of parallel accesses*) decreases if, for example, an error occurs in the system journal file. To prevent journal I/O performance from declining as a result of having too few element files, you can specify a minimum guaranteed number of parallel access files. This is known as the *minimum dispersed files for parallel access*. You can also specify the *maximum dispersed files for parallel access*, which indicates the maximum number of element files that can be accessed in parallel.

Specify the element file names when specifying a filegroup in the system journal service definition. Specify the maximum dispersed files for parallel access in the `jnl_max_file_dispersion` operand and the minimum dispersed files for parallel access in the `jnl_min_file_dispersion` operand.

## (5) Types of journals acquired

### (a) Contents of synchronization point journals

At a complete recovery or UAP partial recovery, the consistency of resources must be assured by performing commit or rollback operations for any undetermined transactions. For such commit or rollback operations, OpenTP1 obtains information about whether a transaction determination is completed or how far the processing at a synchronization point had proceeded. This history information is called a *synchronization point journal* and is used to decide whether OpenTP1 must perform recovery operations for the resource. The synchronization point journals are named BJ, DJ, HJ, PJ, and TJ. They are described in the following table.

Table 4-5: Synchronization point journals

Type	Description
PJ	A journal output when the root transaction branch or a transaction branch starts commit processing.
HJ	A journal output when a transaction branch becomes ready to start commit processing. This journal is not output for the root transaction branch.
BJ	A journal output when a transaction is rolled back. This journal is output for the root transaction branch and transaction branches.
TJ	A journal output when synchronization point processing of a transaction is terminated. This journal is output for the root transaction branch and transaction branches.
DJ	A journal output when the heuristic decision of a transaction is performed. This journal is output for the root transaction branch and transaction branches.

### (b) Contents of recovery journals

In a complete recovery or partial recovery, OpenTP1 first examines the synchronization point journal to determine whether any resources must be recovered. If resources must be recovered, OpenTP1 first recovers databases and system and other

tables. OpenTP1 obtains the necessary update information for the resources from the recovery journals. The recovery journals are named CJ and FJ. They are described in the following table.

*Table 4-6: Recovery journals*

Type	Description
FJ	Consists of update information for DAM files. DAM files are recovered using this FJ recovery journal during a complete recovery or partial recovery.
CJ	Consist of update information for recovery target tables managed by the MCF and TAM services. The CJ recovery journal is obtained, for example, at a synchronization point. The complete recovery of a table uses this CJ recovery journal.
	Consist of update information for recovery target tables managed by the transaction management service. The CJ recovery journal is obtained, for example, at a synchronization point. Entire recovery of a table during a complete recovery uses this CJ recovery journal.

### (c) Contents of statistical journals

Statistical journals contain information for tuning the system. Statistical journals (AJ, GJ, IJ, OJ, MJ, and SJ) are necessary to tune the system.

Of the statistical journals, information for AJ, GJ, IJ, MJ, and OJ is obtained only when OpenTP1 uses MCF to exchange messages with another system. You can use commands to start (`mcf_tactmj` command) and end (`mcf_tdcctmj` command) obtaining information for MJ. In the application attribute definition for each application of an MHP, you can define whether or not to obtain information for GJ, IJ, and OJ. In the logical terminal definition for each logical terminal, you can specify whether to obtain information for AJ.

SJ contains statistics for the entire OpenTP1 system: these statistics include system statistics and transaction statistics. The system statistics are obtained at a user-specified interval after entering the command `dcstats`. In the transaction service definition, you can define whether to obtain the transaction statistics. Statistical journals are listed in the following table.

*Table 4-7: Statistical journals*

Type	Description
MJ	Message input information before editing the input, and message output information after editing the output.
IJ	Message input information obtained immediately before a received message is stored in the input queue.
GJ	MHP receive information obtained immediately before MHP receives a message with the segment-receiving function (i.e., with the <code>dc_mcf_receive()</code> C function or the equivalent COBOL subprogram).

Type	Description
OJ	Message output information obtained immediately after a send message is placed in the output queue.
AJ	Transmission-completion information obtained immediately after a transmission-completed message is received from another system in response to a message sent earlier to that other system.
SJ	Statistics of OpenTP1 activities allowing the user to examine the OpenTP1 operation status. The SJ journal contains both system statistics and transaction statistics. The system statistics cover the system service and UAPs. The status of OpenTP1 can be obtained at set intervals. The transaction statistics cover the processing over transactions. The transaction statistics are obtained when synchronization point processing is completed for each transaction branch.

#### (d) Contents of user journals

OpenTP1 provides the `dc_jnl_ujput()` function to enable a user journal to be stored in a system journal file. The user journal (UJ) is used for system tuning or for collecting information about the user's application. The user journal can be output inside or outside a transaction.

For details about how to obtain a user journal, see the *OpenTP1 Programming Guide*.

#### (6) Status of system journal files

The filegroup of system journal files is either available or unavailable for journal output:

- Available

The number of opened physical files in a filegroup is greater than the required number. The required number of physical files is determined according to the value specified in the system journal service definition. Available filegroups are *current* or *standby*.

- Unavailable

The number of opened physical files in a filegroup is less than the required number. Unavailable filegroups are also called *reserved*.

Filegroups are managed as having *current* or *standby* status if available for journal output, or in *reserved* status if unavailable for journal output. The required number of element files is determined by the value specified in the system journal service definition.

- Current

An available filegroup which is the current destination of journals to be output. There is always only one current filegroup.

- Standby

An available filegroup which is not currently the destination of journals to be

output but is waiting to be switched to become current. The two standby statuses are:

- **Ready to be the next swap destination:**  
A standby filegroup that can be overwritten (there is no journal that is required for system recovery) and is already unloaded.
- **Not ready to be the next swap destination:**  
A standby filegroup that cannot be overwritten or is waiting to be unloaded.
- **Reserved**  
An unavailable filegroup.

A system journal file requires at least two filegroups other than the reserved filegroups.

When OpenTP1 starts normally, all the filegroups specified as ONL, among the filegroups specified in the system journal file service definition, are opened. Among the opened filegroups, the first filegroup that is specified becomes the current filegroup; other filegroups become the standby filegroups. A filegroup that has fewer physical files than the required number or a filegroup that is not specified as ONL becomes a reserved filegroup.

In a system journal file, journals are always output to the current filegroup. If the current filegroup becomes full, one of the standby filegroups becomes the current filegroup. The previous current filegroup becomes one of the standby filegroups. If all the available filegroups become full, journals are again output to the first filegroup.

### **(7) Unloading system journals to a file**

The journals in a standby system journal file must be copied to another file. This copying is called *unloading*. To unload a system journal file, use the `jnlunlfg` command. Alternatively, specify `Y` in the `jnl_auto_unload` operand of the system journal service definition to enable automatic unloading. The file to which a standby system journal file is unloaded is called an *unloaded-journals file* (or, in some manual versions, an *unload journal file*).

In an *unload check*, OpenTP1 checks whether or not the journals in a logical filegroup have been unloaded. If the journals have not been unloaded yet, OpenTP1 does not swap the filegroup with the current filegroup. If there is no swappable standby system file, OpenTP1 terminates abnormally. For the processing in this case, see 5.3 *Failure and error recovery*.

#### **(a) Restraining an unload check**

The standby filegroup, which is usually in the unload wait status, can be specified to be used as is. This is called an *unload check restraint*. When restraining an unload check, specify `N` in the `jnl_unload_check` operand in the system journal service definition.

If the unload check is restrained, do not unload the journal files using a command during the operation of OpenTP1. If an unload is performed by specifying an unload check restraint, unload the file after the concerned files are once closed by the `jnlclsfg` command.

When the unload check is restrained, neither journal edit command `jnlcolc` command nor `jnlstts` command can be executed as input of an unload journal file, because the unload journal file cannot be created.

If an unload check is restrained, the unload wait status may be output when confirming the filegroup status using the `jnlis` command. Even if the unload wait status is output, the current filegroups are actually used.

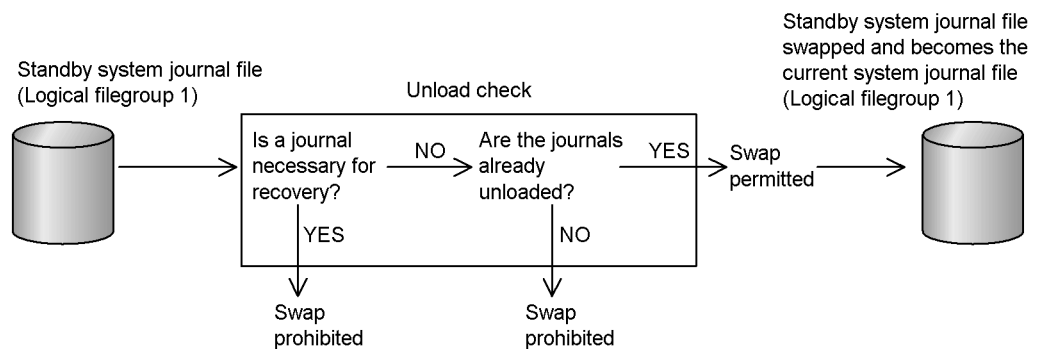
### (8) Swapping system journal files

In a system journal swap the current logical filegroup becomes a standby logical filegroup, and a standby logical filegroup becomes the current logical filegroup. Not all standby logical filegroups are eligible to be swapped in as the current filegroup. A standby filegroup whose journals have not been unloaded yet or a standby filegroup that contains a journal necessary for recovery cannot be swapped with the current filegroup. See 4.2.3(4) *Multigeneration Guarantee facility* for details about checking whether a system journal filegroup contains a journal necessary for recovery.

If there are no swappable files, OpenTP1 terminates abnormally. For the processing in this case, see 5.3 *Failure and error recovery*.

Figure 4-9 shows the conditions on swapping system journal filegroups.

Figure 4-9: Swapping system journal filegroups



### (9) Opening reserved filegroups at complete recovery

When performing a complete recovery after an error occurs during online processing, OpenTP1 outputs journals that are used for transaction determinations. In such a case, if the opened system journal filegroups cannot store more journal information, OpenTP1 terminates abnormally again.



If there are too few journal filegroups opened at complete recovery, reserved filegroups can be opened to store the journals. In the system journal service definition you can define whether or not reserved filegroups are to be opened. If there is a swappable system journal filegroup that can be overwritten after its contents are unloaded, reserved filegroups are not opened. Unload the journal by the command `jnlunlfg`.

### 4.2.3 System files: checkpoint dump files

#### (1) Purpose of checkpoint dump files

Using only the recovery journals to recover the online system from a stoppage would require all the journals from the start of the online processing, and using all these journals might take a lot of time. This recovery time can be reduced by requesting OpenTP1 to periodically save the status of system tables needed in recovery. OpenTP1 then saves this information at various *checkpoints*.

The table information obtained at a checkpoint is called a *checkpoint dump*. In a complete recovery of an online system OpenTP1 does not need to use all the recovery journals from the beginning of processing: OpenTP1 can use checkpoint dumps and only those recovery journals obtained from the time the last checkpoint dump was obtained until the time the online system stopped.

Since OpenTP1 performs recovery for each system service, it obtains a checkpoint dump for each system service for which tables in memory need to be recovered: the transaction service and MCF service. A file is allocated to each system service to store the checkpoint dumps. This file is called a *checkpoint dump file*.

If you have specified that checkpoint dumps are to be taken, the dumps are obtained at these checkpoints:

- when a system journal file is swapped
- when all the journals specified in the system journal service definition are stored in the system journal file
- when OpenTP1 starts or terminates (for the transaction service only)
- when MCF starts or terminates (for the MCF service only)

#### (2) Structure of checkpoint dump files

A checkpoint dump file is a *logical filegroup*, and the actual file entity that obtains a checkpoint dump is called a *physical file*. A filegroup consists of one or two physical files. A filegroup that consists of two physical files is called a *duplicated checkpoint dump file*. When a checkpoint dump file is duplicated, one physical file is called *system A* and the other physical file is called *system B*.

A filegroup that consists of two physical files is called a *duplicated checkpoint dump file*. When a checkpoint dump file is duplicated, one physical file is called *system A*

and the other physical file is called *system B*.

For each of the system services that require checkpoint dump files, you must prepare a checkpoint dump service definition, specifying the system service name, filegroup, and physical file relationships.

The filegroup defined in the checkpoint dump service definition can be given any name, which will be used when working with the checkpoint dump file.

The filegroup in a checkpoint dump file has one of the following statuses:

- Overwrite-permitted, or being written

The filegroup does not contain a checkpoint that would be needed for a complete recovery of the OpenTP1 system. It is therefore open and can be overwritten, or is currently being used to collect a checkpoint dump.

- Overwrite-prohibited

The filegroup contains a checkpoint that would be needed for a complete recovery of the OpenTP1 system. It is therefore open, but overwriting is prohibited.

- Reserved

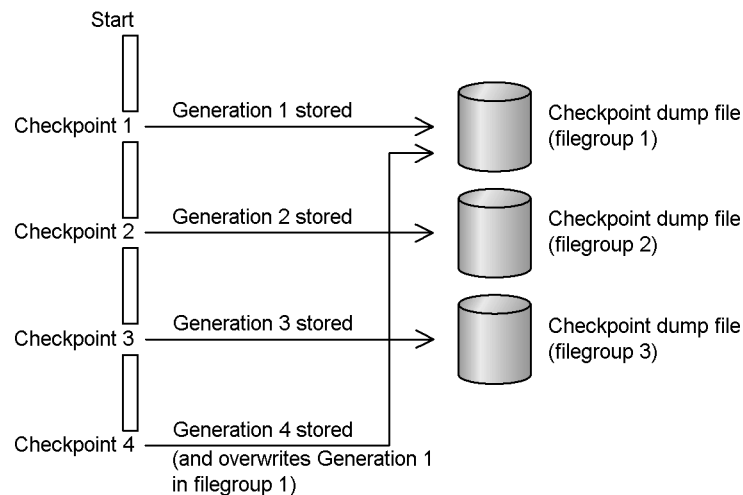
The filegroup is closed and cannot be used online unless opened. This status also applies when the filegroup is defined but has no physical files.

In the checkpoint dump service definition, define the filegroup that is to be open when OpenTP1 starts. Filegroups not defined to be open are placed in reserved status when OpenTP1 starts.

A checkpoint dump taken at one point in time is called a *generation*. Because one generation is stored in one checkpoint dump filegroup, a different filegroup needs to be swapped in for each generation. When checkpoint dumps have been output to all the available filegroups, the data in the first filegroup is overwritten. This method of storing data in multiple filegroups, overwriting each in turn, is called the *round-robin method*. Normally, the filegroup with the most recent generation has *overwrite-prohibited* status, while the other filegroups are placed in *overwrite-permitted* status. However, when using the multi-generation guarantee facility, the filegroups for the two most recent generations are both placed in *overwrite-prohibited* status.

Figure 4-10 shows how checkpoint dumps are assigned to filegroups by round-robin scheduling.

*Figure 4-10: Most recent checkpoint dump generation overwrites earlier checkpoint-dump generation*



### (3) Adding physical files for checkpoint dump files

If an error occurs in a checkpoint dump file during online processing or if there are insufficient files for operations, physical files can be added dynamically. The method to dynamically add such physical files differs according to whether a reserved file has been defined.

#### (a) Method for adding undefined physical files during online processing

Even with physical files that have not been defined in the checkpoint dump service definition, you can use commands to add, during online processing, the physical files to the files used for checkpoint processing.

#### (b) Method for adding standby files

When a standby file has been previously defined in the checkpoint dump service definition, files can be assigned by opening the file with the `jnlpnfg` command or opening the file automatically whenever OpenTP1 starts.

Using the automatic open facility enables automatic allocation of the standby file for the current file when the number of online physical files decreases to the number of guaranteed-valid generations (described below). You can specify automatic opening in the checkpoint dump service definition.

#### (c) Method for deleting physical files for checkpoint dump files

Physical files that are added dynamically to a reserved filegroup can be deleted using the `jnlde1pf` command.

#### **(4) Multigeneration Guarantee facility**

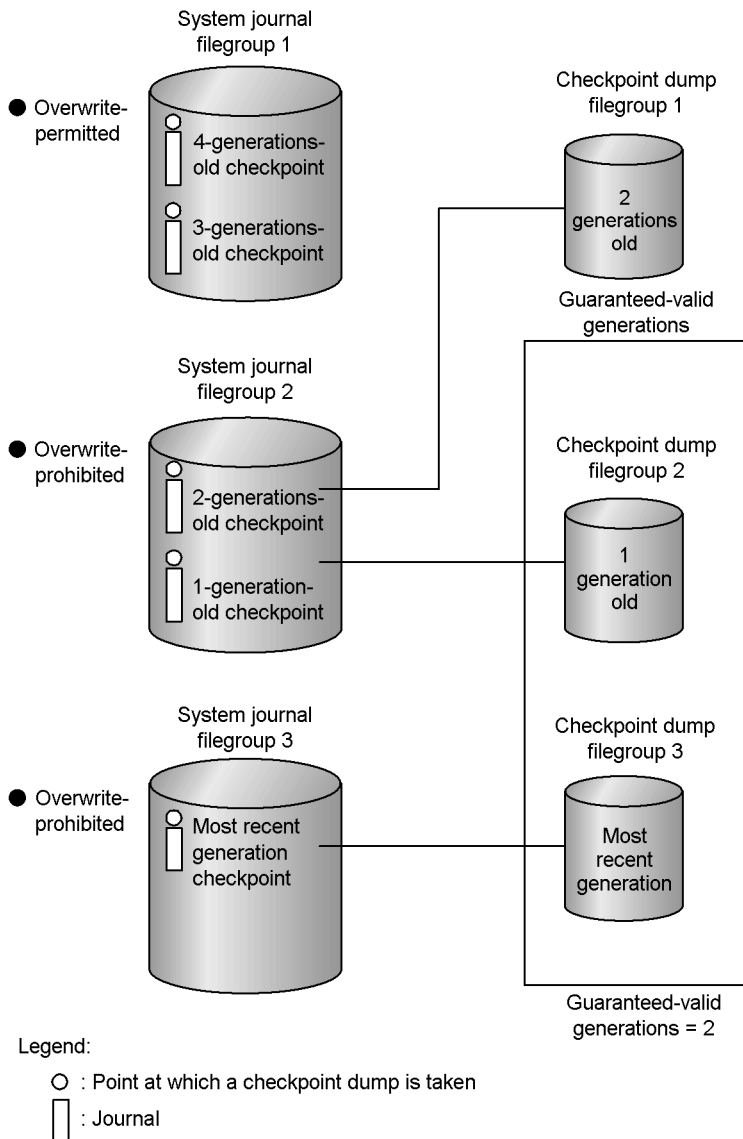
The *Multigeneration Guarantee facility* improves OpenTP1 reliability by enabling OpenTP1 to recover in situations where the filegroup storing the most recent checkpoint-dump generation cannot be read for some reason. In such a case, OpenTP1 can recover by reading the filegroup that contains the generation preceding the most recent one. The Multigeneration Guarantee facility thus prevents the filegroups containing the last two checkpoint-dump generations from being overwritten. The overwrite-prohibited generations are called *guaranteed-valid generations* (or, in some manual versions, *valid guarantee generations*). The number of guaranteed-valid generations is 2 when the Multigeneration Guarantee facility is used and 1 when not used.

OpenTP1 suppresses the overwriting of the system journal filegroups used to store the guaranteed-valid generation of checkpoint dumps that are required for recovery. In an *overwrite check*, OpenTP1 checks whether the system journal file for the guaranteed-valid generation can be overwritten.

##### **(a) Recovery when using the Multigeneration Guarantee facility**

When the multi-generation guarantee facility is enabled, the filegroup containing the most recent checkpoint generation is read first. Recovery processing is then performed, based on all journals collected since the most recent generation. If an error occurs for any reason and the filegroup containing the most recent generation cannot be read, the checkpoint dump file for the preceding generation is read. Recovery processing in this case is based on all journals collected since the preceding generation. If neither of the filegroups storing guaranteed-valid generations can be read, the next most recent generation is used, although journals earlier than the guaranteed-valid generations may have been overwritten and cannot be recovered. The required number of checkpoint dump filegroups that are online and in a status other than *reserved* is the number of guaranteed-valid generations + 1. The following figure shows the relationships between guaranteed-valid generations and system journal files.

Figure 4-11: Guaranteed-valid generations and system journal files



**(b) Guaranteed-valid generations when using the fallback facility**

Even when the number of filegroups required for online operations or for restarting processing falls below one plus the number of guaranteed-valid generations, processing can still continue if a minimum of 2 filegroups are usable. This feature is called the *fallback facility for checkpoint dumps*. In the checkpoint dump service

definition you can specify whether to use the fallback facility.

Note that there is a drawback to using this fallback facility. When an error occurs in an OpenTP1 system during fallback operations, information for restart is guaranteed for only one filegroup. If that filegroup should fail, recovery is not possible.

When fallback operation occurs, a message informs the user that OpenTP1 has changed from ordinary operation to fallback operation. When such a message is output, the OpenTP1 administrator should quickly prepare a filegroup usable for a guaranteed-valid generation. After the filegroup is prepared, another message informs the administrator that fallback operation has switched to ordinary operation.

#### **(5) Guaranteed-valid generations and journal sizes**

The required size of a system journal file increases in proportion to the number of guaranteed-valid generations for checkpoint dump files. You can estimate the required size (that is, the required number of blocks) of the system journal file as follows:

Number of journal blocks used to store journal information between checkpoints  $\times$  (1 + number of guaranteed-valid generations)

#### **(6) Operation when a checkpoint dump file is duplicated**

When a filegroup is operated with a duplicated checkpoint dump file, OpenTP1 outputs the same checkpoint dump to both systems A and B. If a failure occurs in one system while reading the checkpoint dump, the same data can be read from the other system. In this way, duplicating a checkpoint dump file increases reliability.

##### **(a) Specification in the checkpoint dump service definition**

To duplicate a checkpoint dump file, specify `Y` in the `jnl_dual` option of the checkpoint dump service definition. At this time, specify two physical files (system A and system B) in a filegroup.

You should store the physical file of system A and the physical file of system B on separate disks to prevent a failure in both physical files at the same time. The size of the physical file of system A and that of system B need not be the same. If the sizes are different, however, OpenTP1 considers the smaller file to be the size of the checkpoint dump. To use resources efficiently, match the sizes whenever possible.

OpenTP1 manages the generations of each filegroup. When you use OpenTP1 commands to open and close files, the operation is executed in units of filegroups; therefore, it is impossible to open or close only system A or B during online processing.

When a checkpoint dump file is duplicated, one filegroup requires two physical files. If a failure occurs either in system A or B, the filegroup becomes reserved.

##### **(b) One-system operation of the checkpoint dump file**

You can select whether to enable or disable one-system operation if an error occurs on

either physical file of a checkpoint duplicated dump file.

- When one-system operation is available

As long as the physical file of either system A or system B is open, the file group remains open. If an error occurs on the physical file of either system, the processing is continued with a normal system. This operation can validate the checkpoint dump of the normal system.

- When one-system operation is unavailable

Unless the physical files of both systems A and B are open, the file group is placed in the reserved status. If an error occurs on either physical file, the file group is also placed in the reserved status. If an error occurs when reading data from system A, the file group is placed in reserved status after reading data from system B.

### (c) Notes on dual-system operation

When one-system operation is unavailable, it is impossible to open or close only either system.

When one-system operation is available, it is possible to open or close only either system. However, an overwrite-prohibited file group cannot be closed, whether one-system operation is available or not.

Table 4-8 shows the differences between when one-system operation is available and when unavailable.

*Table 4-8:* Differences between when one-system operation is available and when unavailable

Operation	Mode	
	One-system operation available	One-system operation unavailable
Allocating only one system while online	Possible	Possible
Disconnecting only one system while online	Possible	Possible
Opening only one system	Possible when both systems are allocated.	Impossible
Closing only one system (when overwrite-prohibited)	Impossible	Impossible
Closing only one system (when overwrite-permitted)	Possible	Impossible

## 4.2.4 System files: transaction recovery journal file

### (1) Purpose of the transaction recovery journal file

The transaction recovery journal file is separate from the system journal files and acquires various types of information about transactions at each transaction branch (that is, each UAP process). A transaction recovery journal file is created as a UNIX file with the name of `trfxxxxxxx` (where *x* indicates an integer) in the `$DCDIR/spool/dctjlinf/` directory. Therefore, no transaction recovery journal file can be used in a System Switchover configuration.

In the user service definition for each service group, you can define whether to obtain a transaction recovery journal file. This file overcomes problems such as the following:

- In transaction processing that spans multiple UAP processes, one process taking a long time might cause multiple system journal files to obtain journal information for the resources updated in a transaction. If online processing stops during transaction processing, all those system journal files must be read to recover the resources. If all such files must be read, recovery will take a long time.
- A system journal file that is being used cannot be swapped so it is possible that, depending on the size and number of the system journal files, before the end of a transaction's processing there might be no system journal files that can be swapped. If no system journal files can be swapped, OpenTP1 terminates abnormally.

To avoid the above problems, you can use the user service definition to specify a special file in which OpenTP1 stores the journal (i.e., information) about a transaction that takes a long time. This special file is a *transaction recovery journal file* which is separate from the system journal files. By creating a transaction recovery journal file, you can reduce the number of journals that are read at recovery. Also, even when processing of a transaction takes a long time, system journal files can be swapped and checkpoint dumps can be effective. These results can reduce the time required for recovery at a restart.

Using a separate transaction recovery journal file also helps to prevent those OpenTP1-system abnormal terminations caused by long transaction processing times producing overflows of journal information. We recommend that you use a transaction recovery journal file for UAPs that have long processing times. When you use a transaction recovery journal file, however, the same journal is collected in both the system journal file and the transaction recovery journal file. This might slow down transaction processing.

### (2) Recovering from an error in a transaction recovery journal file

When an error occurs in a transaction recovery journal file during restart processing, OpenTP1 outputs a message to report this situation. When this message is output, use



the command `jnlmkrf` to recover the transaction recovery journal file. Use the `unloaded-journals` file to recover the transaction recovery journal file.

## 4.2.5 System files: server recovery journal file

### (1) Purpose of the server recovery journal file

The server recovery journal file is separate from the system journal files and collects various types of journal information about each system service. Using the server recovery journal file enables each system service to read journals independently, reducing recovery time at a complete-recovery restart.

### (2) Creating the server recovery journal file

OpenTP1 automatically obtains information for the server recovery journal file, which has the filename `xxx_nn` (where `xxx` is the name of the system service and `nn` is an integer) in the directory `$DCDIR/spool/dcsj1/`.

### (3) Recovering from an error in a server recovery journal file

When an error occurs in the server recovery journal file during restart processing, OpenTP1 outputs a message to report this situation. When this message is output, use the command `jnlmkrf` to recover the server recovery journal file. Use the `unloaded-journals` file to recover the server recovery journal file.

## 4.2.6 System files: archive journal files

### (1) Purpose of archive journal files

When using OpenTP1 in a cluster system or parallel-processing system, OpenTP1 provides the *Global Archive Journal facility* to simplify the unloading of system journal files from multiple nodes. Rather than having to unload the standby system journal file for each individual node, a user can:

Use the OpenTP1 Global Archive Journal facility to automatically copy the standby system journal files from a number of nodes into an *archive journal file* on a single dedicated node.

(For details on the Global Archive Journal facility, see 6.2.3 *Global Archive Journal facility*.)

1. Use the command `jnlunlfg` to unload the contents of the archive journal file to a file called the global archive unloaded-journals file.

For global archive journals, you can select the kind of the journal to be archived. Specify the kind of the journal record by the `jnl_arc_rec_kind` operand in the system journal definition. For details of the kinds of journals, see 4.2.2 *System files: system journal files*.

### (2) Structure of archive journal files

Up to 16 types of archive journal files can be created. The created archive journal files

are within the archive-journal target node, which is the node used for archiving. As with system journal files, individual archive journal files are logical filegroups containing physical files.

Before using an archive journal file, you must specify whether the file is to be opened when OpenTP1 starts. By specifying the filegroup name in a command parameter (e.g., in `jnlpnfg` or `jnl1s`) an archive journal file can be operated on in logical filegroup units.

Archive journal files can be grouped into *resource groups*. A resource group is a group of archive journal files that are separated by intended use. The global archive journal service uses the names of resource groups to manage archive journal files. Up to 20 nodes can be archived in one resource group.

### **(3) Creating archive journal files**

To create an archive journal file, create a physical file in the OpenTP1 file system using the `jnlinit` command, and specify the archive journal file in the archive journal service definition.

In the archive journal service definition where the resource group is to be defined, specify the filegroup name, physical file name, and element file name.

- Physical file name  
Pathname that indicates the real OpenTP1 file.
- Element file name  
A name given to the element file when using the parallel access facility for archive journal files.

An element file can be duplicated. A filegroup name is a name given to a group of one or more element files. When not using the parallel access facility for archive journal files (when there is only one element file in the filegroup), the element file name can be omitted. Up to 8 element files can be contained in a filegroup.

Before using an archive journal file, specify, for each filegroup, whether to open the archive journal file at the same time OpenTP1 starts. By specifying a filegroup name in a command argument, the archive journal file can be operated in logical filegroup units. Note that one resource group requires two or more filegroups.

### **(4) Parallel access facility for archive journal files**

An archive journal file can be used by configuring a filegroup using multiple element files. This facility is called the *parallel access facility for archive journal files*.

In an archive journal file, the journals for multiple nodes are constantly input and output. Sometimes the input and output operations for the archive journal file cannot keep up with the journals that need to be archived. In this situation, configure a filegroup using more than one element file to access the archive journal file in parallel.

This increases the performance of the archive.

Physical files that make up a filegroup can use the same SCSI interface and hard disk; however, when they do, the SCSI interface and the hard disk cannot be accessed in parallel, nor can they provide the full parallel access facility. Try to use different SCSI interfaces and hard disks. The sizes of physical files need not be the same. If the sizes are different, however, OpenTP1 considers the smaller file to be the size of the filegroup. To use resources effectively, match the sizes of physical files whenever possible.

When using the parallel access facility for archive journal files, the number of element files that can access in parallel (number of parallel accesses) decreases if an error occurs in the archive journal file. To make sure the input and output operations of the archive journal file keep up with the journals that need to be archived, you can specify the minimum number of parallel accesses to be guaranteed. This minimum number of parallel accesses to be guaranteed is the minimum number of distributions. The maximum number of distributions is the maximum number of parallel accesses.

When specifying a filegroup in the archive journal service definition, use the element file name. Specify the maximum number of distributions in the `jnl_max_file_dispersion` operand and the minimum number of distributions in the `jnl_min_file_dispersion` operand in the archive journal service definition.

### **(5) Duplicating an archive journal file**

An archive journal file can be duplicated by configuring an element file using two physical files. If a filegroup is operated with the duplicated archive journal file, OpenTP1 outputs the same journal in system A and in system B.

If a failure occurs in one system while reading the checkpoint dump, the same data can be read from the other system. In this way, duplicating a checkpoint dump file increases reliability.

#### **(a) Specifying the archive journal file definition**

To duplicate an archive journal file, specify `Y` in the `jnl_dual` operand in the archive journal file definition. At this time, specify two physical files (system A and system B) in one filegroup. You should store the physical file of system A and the physical file of system B on separate disks to prevent a failure in both physical files at the same time. The size of the physical file of system A and that of system B need not be the same. If the sizes are different, however, OpenTP1 considers the smaller file to be the size of the journal. To use resources efficiently, match the sizes whenever possible.

#### **(b) Single-system operation and non-single-system operation**

When an archive journal file is duplicated, single-system operation or non-single-system operation can be selected.

- Single-system operation

If either of the two physical files in an element file can be used, the element file can be used.

- Non-single-system operation

The element file can be used only when both physical files in the element file can be used.

The archive journal service definition allows or disallows single-system operation. When single-system operation is permitted, even if an error occurs in one physical file, the system continues operating with the remaining physical file until the error is corrected. During single-system operation, the archive journal file is not duplicated, decreasing reliability.

### **(6) Status of archive journal files**

An archive journal file can have two statuses, the same as a system journal file.

- Available

The number of usable element files among the element files in a filegroup is greater than the required number.

- Unavailable

The number of usable element files among the element files in a filegroup is less than the required number.

An *available* element file implies that the required number of physical files is opened in that element file. An element file is *unavailable* if the required number of physical files are not opened in the element file. The required number of element files and the required number of physical files are determined according to the values specified in the archive journal service definition.

Filegroups are managed as having *current* or *standby* status if available for journal output, or in *reserved* status if unavailable for journal output. The required number of element files and physical files are determined by the values specified in the archive journal service definition.

- Current

An available filegroup which is the current destination of journals to be output. There is always only one current filegroup.

- Standby

An available filegroup which is not currently the destination of journals to be output but is waiting to be switched to become current. The two standby statuses are:

- **Ready to be the next swap destination:**

A standby filegroup that can be overwritten (there is no journal that is

required for recovery) and is already unloaded.

- **Not ready to be the next swap destination:**

A standby filegroup that cannot be overwritten or is waiting to be unloaded.

- **Reserved**

An unavailable filegroup.

An archive journal file requires at least two filegroups other than the reserved filegroups.

### **(7) Unloading archive journal files**

When an archive journal file is waiting for an unload, you should use the command `jnlunlfg` to unload. A filegroup of an archive journal file which has not been unloaded might cause a complete lack of swappable files, which would cause an abnormal termination of the global archive journal service.

As with a system journal file, an unloaded ordinary file allows DAM file recovery and editing of files containing operation statistics. Also, a user journal can be inherited by an offline program.

A global archive unloaded-journals file contains system journal information about multiple OpenTP1 nodes. You can edit this journal: for example, by editing merged operation statistics or by sorting multiple OpenTP1 node journals by time.

#### **(a) Restraining an unload check**

The standby filegroup which is usually in the unload wait status can be specified to use as it is. This is called a unload check restraint. When restraining an unload check, specify `N` to the `jnl_unload_check` operand in the system journal service definition. To restrain an unload check, specify `N` in the `jnl_unload_check` operand in the archive journal service definition.

If the unload check is restrained, do not unload the journal files using a command during the operation of OpenTP1. If an unload is performed by specifying an unload check restraint, unload the file after the files are once closed by the `jnlclsfg` command.

When the unload check is restrained, the journal edit commands (`jnlcolc` command, `jnlstts` command) cannot be executed as input for a global archive unload journal file because the global archive unload journal file cannot be created.

If an unload check is restrained, the unload wait status may be output when confirming the filegroup status using the `jnlis` command. Even if the unload wait status is output, the current filegroups are actually used.

### **(8) Status of system journal files**

In the system journal services related to the Global Archive Journal facility, you can

add the following journal file statuses: *archived* or *not archived*. These indicate whether or not a system journal filegroup has been archived. You can display this status of a system journal file with the `journal` command.

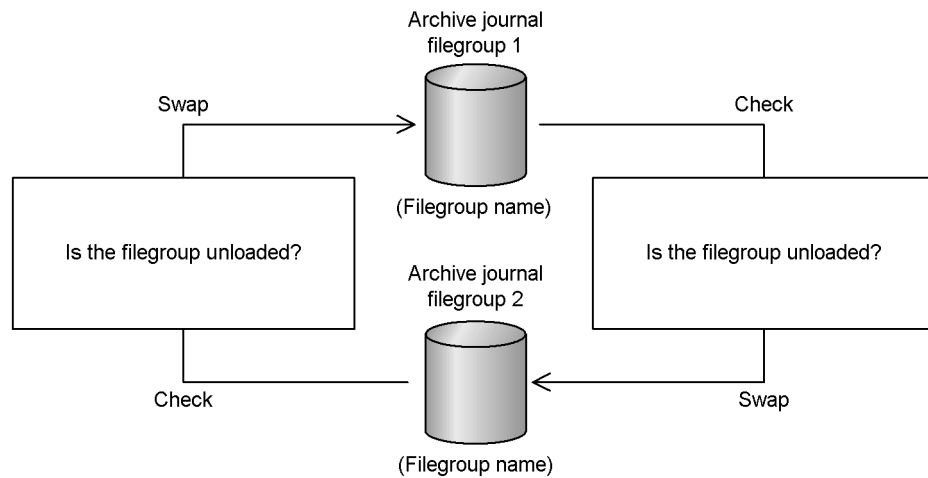
After a system journal filegroup is copied to an archive journal file, the system journal filegroup is eligible to be swapped even if the archive journal file has not yet been unloaded.

### (9) Swapping archive journal files

Some standby archive journal filegroups can become swap destinations while others cannot. Those that can become swap destinations are filegroups that are already unloaded. Filegroups that are not unloaded cannot be swap destinations. If there is no swap destination for the archive journal file, OpenTP1 terminates abnormally: see 5.3.3(1)(d) *Recovering from system-file errors: archive journal files*.

Figure 4-12 shows the swapping of archive journal files.

Figure 4-12: Swapping archive journal files



---

## 4.3 Queue files

---

The following OpenTP1 files are used by the message exchange and message queuing facilities:

- Queue files: MCF message queue file  
Used for message exchange (communication using TP1/Message Control).
- Queue files: MQA message queue file  
Used for message queuing (communication using TP1/Message Queue).

### 4.3.1 Queue files: MCF message queue file

#### (1) Purpose of the MCF message queue file

The *MCF message queue file* is a file in the OpenTP1 file system and is used when MCF exchanges messages with another system. The MCF message queue file is used as a disk queue for I/O messages: i.e., messages received or messages to be sent. The input queue is used for managing received messages such as requests, and the output queue is used for managing messages to be sent such as processing request results.

#### (2) Structure of the MCF message queue file

OpenTP1 handles MCF message queue files in a logical unit called a *queue group*, while the actual file that obtains I/O messages is called a *physical file*. A queue group always consists of one physical file. For each queue, you must create an MCF message queue service definition in which the correspondence between a queue group and a physical file is defined. In the MCF message queue service definition, you can assign any name (ID) to a queue group. Using the *queue group ID* allows MCF message queue file operation in units of queue groups: for example, you can monitor the percentage of physical files that are used.

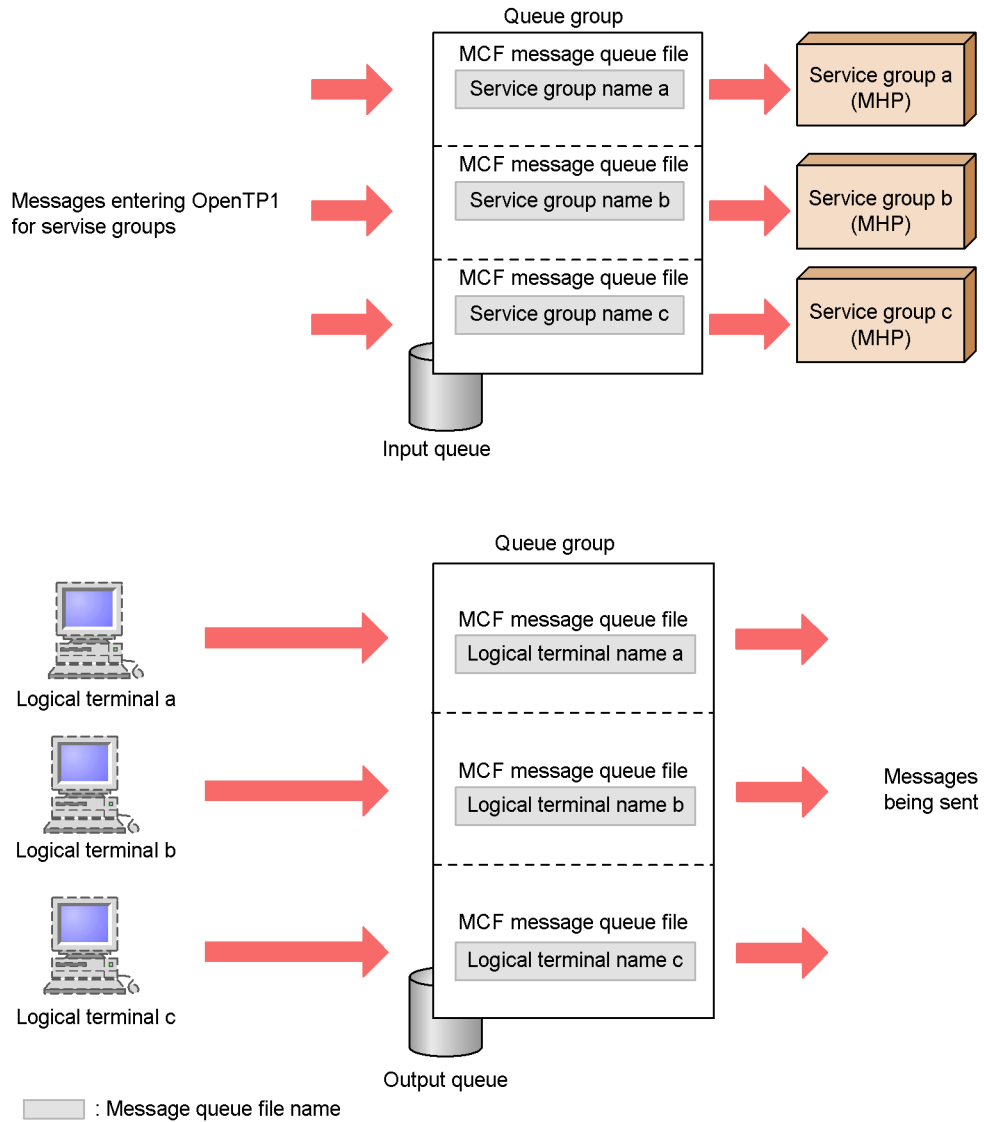
In the message queue service definition, you can assign any name (ID) to a queue group. Using the *queue group ID*, message queue files can be operated as a queue group, when monitoring the usage of physical files, for example.

A queue group can be divided into a number of service groups or logical terminals. Each service group or logical terminal is known as a *queue file*. The name of the service group or logical terminal is called the *queue file name*.

Input and output messages are scheduled on the basis of a particular service group or logical terminal, respectively. Dividing a queue group into separate service groups and logical terminals enhances performance when the same resource is being accessed.

Figure 4-13 shows the relationship between queue groups and MCF message queue files.

Figure 4-13: Queue groups and message queue files



### 4.3.2 Queue files: MQA message queue file

#### (1) Purpose of the MQA message queue file

An MQA message queue file registers the messages in a queue from UAPs. The queue manager (TP1/Message Queue for OpenTP1) sends each message to the appropriate destination. Desired messages can be retrieved regardless of the registration order by



the MQA service definition and the specification of MQI.

For the MQA message queue file configuration, types of queues, and operation procedures, see the *OpenTP1 TP1/Message Queue User's Guide*.

---

## 4.4 User data files

---

This section describes the user data files (user files) used in OpenTP1 application processing. Three types of user files are used in OpenTP1:

- DAM files  
Used as OpenTP1-specific direct access files.
- TAM files  
Used as OpenTP1-specific direct access files that can be rapidly accessed using the table access method.
- ISAM files  
Used as indexed sequential access files that conform to the X/Open ISAM model. For details about ISAM files, see the manual *Indexed Sequential Access Method ISAM*.

All OpenTP1 user files are created on an OpenTP1 file.

In addition to these three file types, user files managed by the IST service (which accesses tables in shared memory) or by a database management system (DBMS) can be used in OpenTP1.

### 4.4.1 User files: DAM files (TP1/FS/Direct Access)

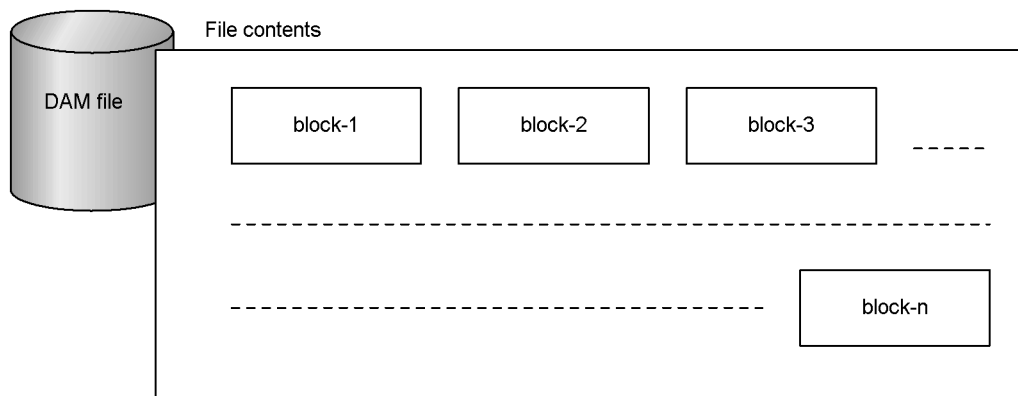
This section describes DAM (direct access method) files, which are one of three types of user files used in OpenTP1 application processing. The three types of user files are DAM, TAM, and ISAM. DAM files are created as OpenTP1 files.

The OpenTP1 DAM file service enables OpenTP1 files in an OpenTP1 file system to be accessed as direct-organization files, enables OpenTP1 to manage the access as part of a transaction, and enables OpenTP1 to manage the file status. To use the DAM files in OpenTP1, *TP1/FS/Direct Access* is needed.

You use the DAM service definition when creating a DAM file. In the definition you must specify the *name of a physical file* and the *name of a logical file* for the DAM file. The name of the physical file, which is the complete pathname for an OpenTP1 file entity, is made up of the directory names and the filename. Each name of a logical file matches the name of a corresponding physical file; the name of the logical file is for the convenience of the user and is assigned by the user. Using the logical file name allows the user to use the DAM file without knowledge of the structure of the physical file.

Figure 4-14 shows the configuration of the DAM file.

Figure 4-14: DAM file configuration



**(1) Features of a DAM file**

Table 4-9 lists the special features of DAM files.

Table 4-9: Special features of DAM files

Item	Special feature
File structure	Direct organization file (an OpenTP1 file).
Record format	Fixed length, non-blocked format.
File access	File can be referenced and updated by the direct access method using the relative block number. Consecutive blocks can be input or output all together.
Relation to transaction	The contents of a block that was updated during transaction processing are updated in the file when a commit operation is performed on a transaction or when the maximum number of blocks specified in the DAM service definition is reached. If the transaction is rolled back, the status before the start of transaction is restored.
File backup and restore	The DAM file management command <code>dambackup</code> can backup a DAM file to another file. Also, the most recent contents of a DAM file can be recovered by using the backup file and the recovery journals after the backup was obtained. Recovery of a DAM file uses the DAM FRC facility.
File addition	The DAM file management command <code>damadd</code> can add DAM files into an online system during OpenTP1 operation.
File separation	The DAM file management command <code>damrm</code> can separate the DAM file used in online processing from the operating OpenTP1.
File deletion	The DAM file management command <code>damdel</code> can delete a DAM file separated from OpenTP1.

## **(2) Creating a DAM file**

OpenTP1 files allocated to the OpenTP1 file system are called *physical files*. Offline DAM files are accessed via physical files. Online DAM files are accessed via a logical file that corresponds to a physical file. In the DAM service definition, you specify which physical file corresponds to which logical file in a DAM file.

DAM files should be created after creating an OpenTP1 file system. There are two ways to create DAM files:

- use a command

With this method, you use the DAM management command `damload` to allocate physical files, then output initial data to the physical file.

- use UAPs

With this method, a UAP issues a DAM access function `dc_dam_create()` or `dc_dam_put()` (for file allocation, and outputting initial data) in an offline environment to create the DAM file.

## **(3) I/O functions for a DAM file**

OpenTP1 UAPs can use functions to access DAM files. For details on the DAM file service that can be used by UAPs see the *OpenTP1 Programming Guide*.

## **(4) Locks on a DAM file**

### **(a) Lock granularity**

A lock on a DAM file takes effect within a transaction branch or within a global transaction. In the DAM service definition, you can change whether the lock applies to a transaction branch or to a global transaction.

When a lock applies to a global transaction, an error is not returned even if multiple transaction branches in the same global transaction access the same block or the same file.

For parallel processing of access to DAM files in each transaction branch, you should specify locks that apply to transaction branches because specifying locks that apply to a global transaction can reduce transaction efficiency. When a lock applies to a global transaction, even if DAM files are accessed for each transaction branch, parallel access is not possible and access is by sequential access.

For details on locking, see the *OpenTP1 Programming Guide*.

### **(b) Transactions and lock granularity**

A lock on a DAM file takes effect within a transaction branch or within a global transaction. In the DAM service definition, you can change whether the lock applies to a transaction branch or to a global transaction.

When a lock applies to a global transaction, an error is not returned even if multiple

transaction branches in the same global transaction access the same block or the same file.

For parallel processing of access to DAM files in each transaction branch, you should specify locks that apply to transaction branches because specifying locks that apply to a global transaction can reduce transaction efficiency. When a lock applies to a global transaction, even if DAM files are accessed for each transaction branch, parallel access is not possible and access is by sequential access.

For details on locking, see the *OpenTP1 Programming Guide*.

### **(c) Setting lock release for resources**

When an attempt is made to update, or to reference for an update, a block of an opened DAM file and that block is already locked, you can use a function parameter (e.g., in `dc_dam_open()`, `dc_dam_read()`, and `dc_dam_write()`) to specify whether to wait for the release of the lock. In the lock service definition you can specify how long to wait for the release of the lock. If the lock is not released within this specified period, an error is returned.

## **(5) Deferred update of a DAM file**

In an OpenTP1 *deferred update*, a block of a DAM file that is updated by a transaction is not updated when the synchronization point is reached; the DAM file is updated asynchronously (though note the exception in the next paragraph). A DAM file with a deferred update specification is actually updated from a process used exclusively for output and which is started after some specified period. Specifying a deferred update for a DAM file can reduce the number of I/O operations, which can improve throughput of transactions per unit of time.

Note, however, that when one transaction outputs updates for both a DAM file that has a deferred update specification and a DAM file that receives an ordinary update, both DAM files are updated at a synchronization point.

### **(a) How to specify deferred update for a DAM file**

In the DAM service definition for each DAM file, you can specify whether or not a DAM file is to have a deferred update.

Before the execution of the process used exclusively for output, in the DAM service definition you must specify the buffer area for storing the block and the period for executing the update from the process used exclusively for output. If this execution period is too long or the value for the buffer area is too small, a buffer area overflow might occur or the system might be forced to wait for free area.

When using the DAM service definition to specify a deferred update, you should take care when calculating the values for the buffer area and the time period.

**(b) Caution required when using DAM files that have deferred update specified**

When OpenTP1 abnormally terminates because of a system failure, a DAM file that has a deferred update specified might not contain the results of a transaction that completed before the failure. To guarantee the results of the transaction update of the DAM file, you must always normally terminate OpenTP1 after OpenTP1 is restarted in a complete recovery.

Even for transactions that were completed before the failure, if the processing of the process used exclusively for output was not completed before the failure, a rollback might occur after a complete recovery. If you do not want to rollback a transaction that was completed before the failure, do not specify a deferred update for the DAM file to be accessed.

When a disk error occurs in a DAM file that has a deferred update specification, you should always execute the DAM FRC (file recovery).

**(6) Online backup of a DAM file**

DAM files can be backed up during OpenTP1 online operations. To allow *online backups* of DAM files, execute the `dambkup` command with the `-o` option. If you do not specify the `-o` option, backup is done offline.

To perform an offline backup of a DAM file:

1. Execute the `damhold` command.  
The logical file is logically shut down.
2. Execute the `damrm` command.  
The logically shutdown logical file is separated from online processing.
3. Execute the `dambkup` command without the `-o` option.  
The DAM file is backed up.

Recovery of a DAM file by a file that has been backed up online requires the use of fewer unloaded-journals files. This results in shorter recovery processing times for DAM files when compared to recovering a DAM file that uses backup files for which the `-o` options was not specified.

**(7) Extraction of user data**

You can extract user data from a DAM file without the management information. You can use the `damload` command to allocate the extracted user data as the initial data for a DAM file. However, you cannot use the `damrstx` command to restore the extracted user data. To extract user data, execute the `dambkup` command with the `-d` option.

**(8) Block length extension facility**

When you restore a file that was backed up using the `dambkup` command, you can

extend the block length. This feature is called the *block length extension facility*. By using this facility, you can move data between DAM files with different block lengths.

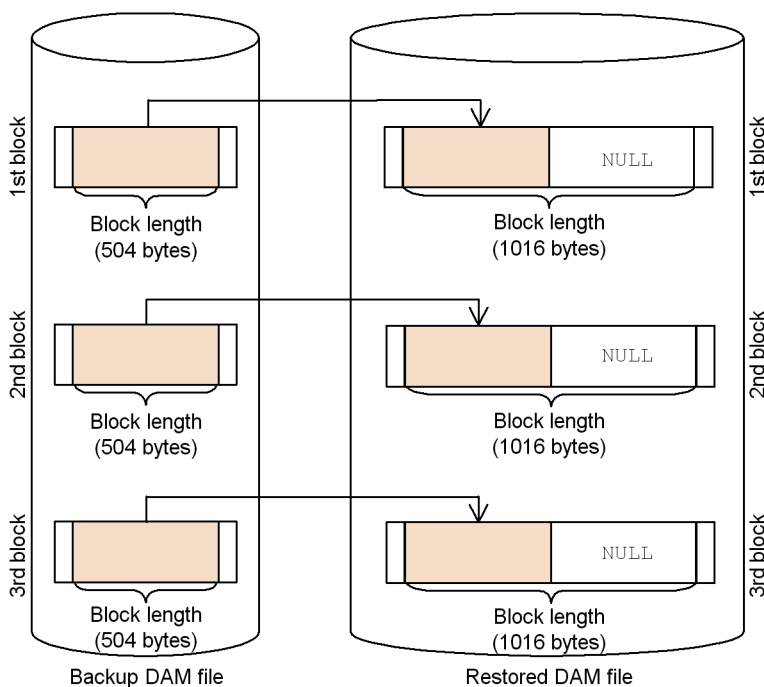
There are two modes for the block length extension facility: a mode that maintains the block configuration of the backup DAM file and a mode that does not maintain the block configuration of the backup DAM file. See below for details.

- Maintaining the original block configuration

You can maintain the block configuration of the backup DAM file in the restored DAM file. You can specify the new block length using the `-e` option in the `damrstr` command.

Figure 4-15 shows an example of restoring a DAM file with a block length of 504 bytes to a DAM file with a block length of 1016 bytes.

*Figure 4-15: Extending the block length maintaining the original block configuration*



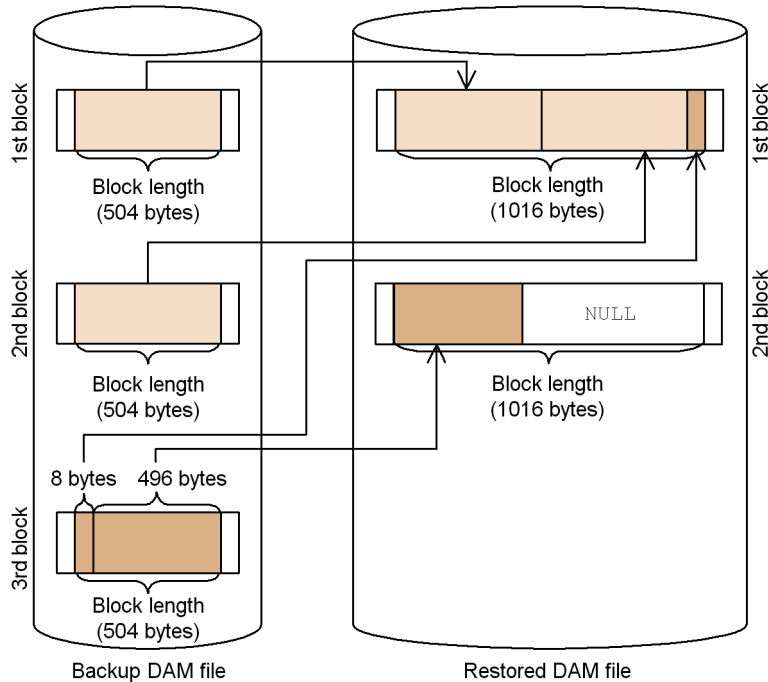
- Not maintaining the original block configuration

You can store the data from the backup DAM file in the restored DAM file from the first block without entering any space. You can specify the new block length using the `-p` option of the `damrstr` command.

Figure 4-16 shows an example of restoring a DAM file with a block length of 504

bytes to a DAM file with a block length of 1016 bytes.

*Figure 4-16:* Extending the block length without maintaining the original block configuration



Note the following when using the block length extension facility.

- You cannot specify a DAM file with an extended block length as a recoverable definition file in the `damfrc` command.
- You cannot extend the block length by using a backup file that is acquired online.

### **(9) Backing up and restoring DAM files using standard input/output**

You can use standard output to the backup destination of a DAM file and standard input to the input file to be restored. Using standard input/output can redirect a command. If you use standard input/output, specify the `-s` option in the `dambackup` and `damrstr` commands, then execute them.

### **(10) Access to unrecoverable DAM files**

DAM files can be created without guaranteeing their consistency and error recovery in a transaction. These DAM files are called *unrecoverable DAM files*. Processing not included in transactions can update and output blocks into the unrecoverable DAM files.



Specify the `-n` option in `damfile`, which is a definition command of the DAM service definition, for an unrecoverable DAM file. If an error occurs while accessing an unrecoverable DAM file, the file data error cannot be recovered.

### (11) Using cache blocks

The DAM service stores the block data in the DAM file it reads in the DAM service-exclusive shared memory. When there are multiple reference requests for the same block, the DAM service returns the block data in the DAM service-exclusive shared memory to the UAP to reduce the number of file I/O operations.

In the DAM service-exclusive shared memory, several pieces of block data are chained and managed for each DAM file. An area in the DAM service-exclusive shared memory storing block data is called a *cache block*.

When a UAP accesses a DAM file, the DAM service secures cache blocks according to the following procedure. This processing is called *cache block securing*.

The procedure for cache block securing is as follows:

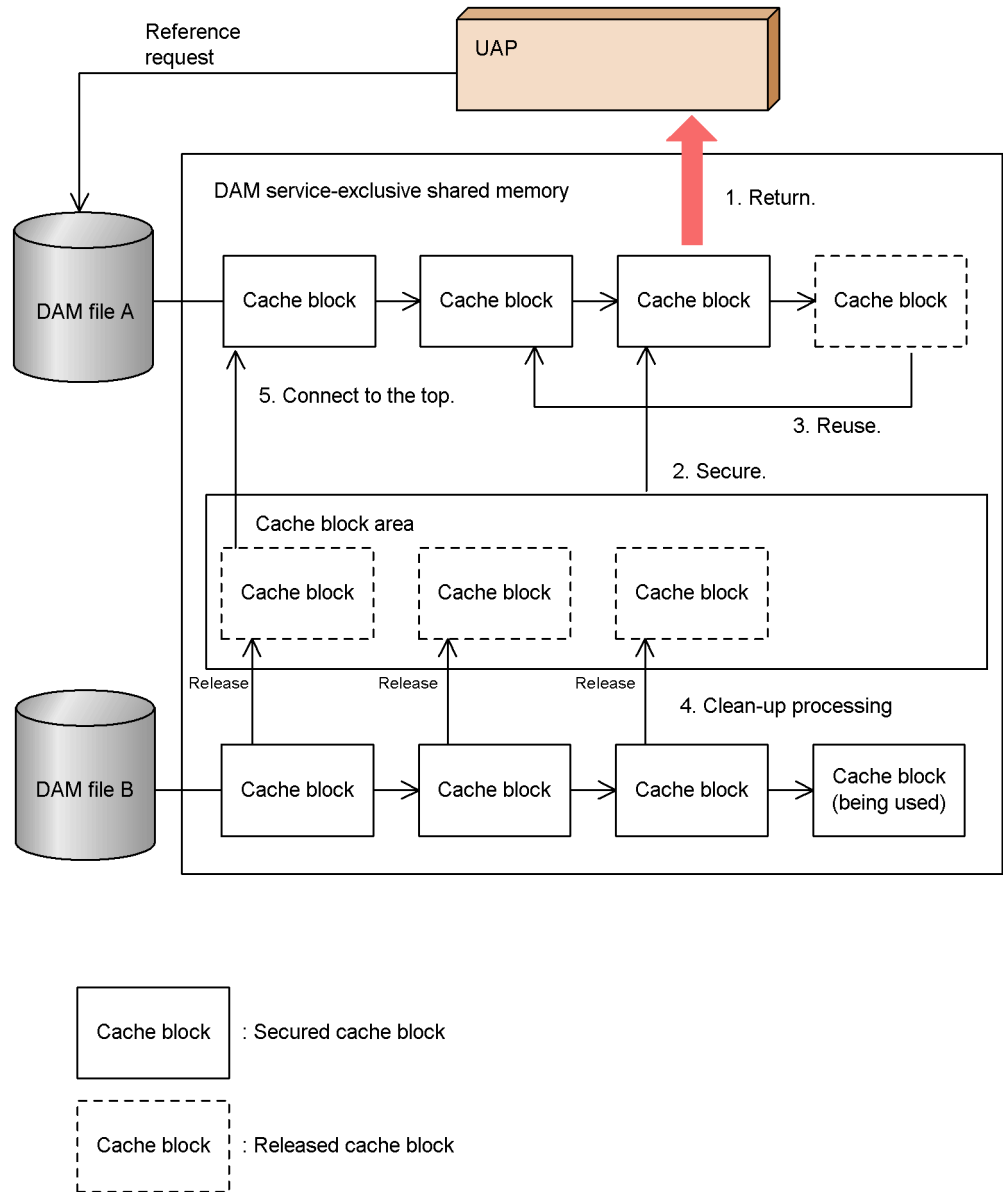
1. The DAM service checks if the cache block corresponding to the required block is connected to the cache block chain for the accessed DAM file. If it is connected, the DAM service returns the data in that cache block to the UAP.
2. If the target cache block is not connected to the cache block chain, the DAM service secures a cache block from the cache block area.
3. If there is no free space in the cache block area, the DAM service reuses a cache block in the cache block chain for the accessed DAM file. In this case, the cache block should not be currently referenced by the transaction being executed. Cache blocks are reused from the oldest cache block in the cache block chain (the last cache block in the cache block chain). The cache blocks to be reused are determined by the value specified in the `dam_cache_reuse_from` operand in the DAM service definition. By default, cache blocks are used from the last cache block.
4. If the DAM service cannot secure a cache block in steps 2 and 3, the DAM service releases all the cache blocks, which are not used by the transaction being executed, in the cache block chain for another DAM file. Then the DAM service secures a cache block from the cache block area. This method of releasing cache blocks is called *clean-up processing*.
5. The DAM service copies the block data of the DAM file to the secured cache block and connects the cache block to the beginning of the cache block chain.

To acquire the number of cache blocks connected to a cache block chain and the total use percentage of the DAM service-exclusive shared memory, use the `damchinf` command.

Figure 4-17 gives an overview of a cache block chain.

Numbers in the figure correspond to the above step numbers.

Figure 4-17: Overview of a cache block chain



**(12) Specifying the threshold for the number of cache blocks**

If a specific DAM file is heavily accessed, the number of cache blocks managed by the

DAM file increases. As a result, searching for and releasing the target cache blocks takes time, resulting in decreased transaction performance.

By specifying the `damch1mt` definition command in the DAM service definition, you can set a limit for the maximum number of cache blocks that can be managed by a single DAM file. This limit is called a *threshold*. Cache blocks are secured up to the threshold. When the threshold is reached, unused cache blocks for the corresponding DAM file are reused and new cache blocks are not secured. If you do not specify a threshold, cache blocks are secured until the shared memory resource is used up.

During online operation you can use `damchdef` command to dynamically change the value to be specified for the `damch1mt` definition command in the DAM service definition.

#### (a) Calculations for cache blocks

You can calculate the size of the area used for cache blocks as follows:

Cache block area size =  $M - (M/576) \times 34$  (bytes)

You can calculate the size of each cache block as follows:

Cache block size =  $(Ab + 8) + 64$  (bytes)

M: Size of the DAM service-exclusive shared memory

Ab: Length of a block in the DAM file to be accessed

#### (b) Examples

Specifying a threshold is effective when a specific DAM file is heavily accessed. The following conditions must be satisfied to specify a threshold:

- The cache memory secured by the DAM service can store 100,000 cache blocks.
- Only `damfileA` is heavily accessed during online operation.
- The number of blocks in `damfileA` is 100,000.
- The search time for a single cache block is 0.1 milliseconds.#
- The file I/O time for a single block is 500 milliseconds.#

#: Depends on the status of the hardware or the process.

A simple value is used for the purpose of explanation.

##### ■ Example 1:

When you do not specify a threshold for the number of cache blocks

If you do not specify a threshold for the number of cache blocks, cache blocks are connected to the cache block chain for `damfileA` as long as the cache block area has free space. In this example, 100,000 cache blocks are connected to the cache

block chain for `damfileA`.

In this case, the chain scan time required for detecting the block data at the end of the cache block chain is  $0.1 \text{ milliseconds} \times 100,000 \text{ blocks} = 10 \text{ seconds}$ . The relative transaction performance is better if direct file I/O is performed. Since `damfileA` is using all the cache blocks that can be secured in the cache memory, clean-up processing will be executed for `damfileA` if you access another DAM file. When clean-up processing is performed, 100,000 cache blocks need to be released, decreasing the transaction performance even more.

■ Example 2:

When you specify 4000 as the threshold for the number of cache blocks

When you specify 4,000 as the threshold for the number of cache blocks, up to 4,000 cache blocks are connected to the cache block chain for `damfileA`. (More than 4,000 cache blocks may be connected depending on the values specified in the DAM service definition.)

In this case, the chain scan time required for detecting the block data at the end of the cache block chain is  $0.1 \text{ milliseconds} \times 4000 \text{ blocks} = 400 \text{ milliseconds}$ . The transaction performance is better than directly performing a file I/O. Since the cache block chain contains only 4000 cache blocks, the remaining 96000 cache blocks are not secured. Therefore, clean-up processing is not performed even if you access another DAM file.

However, since the number of cache blocks connected to the cache block chain for each DAM file is limited, the number of cache blocks in the cache block area decreases, causing some part of the DAM service-exclusive shared memory to be unused.

■ Example 3:

When you specify 0 as the threshold for the number of cache blocks

When you specify 0 as the threshold for the number of cache blocks, the number of cache blocks connected to the cache block chain for `damfileA` depends on the number of blocks that are accessed in the transaction branch and the values set in the DAM service definition.

This example assumes that the following values are set in the DAM service definition:

```
set dam_update_block = 10
set dam_tran_process_count = 5
```

- When all the transaction branches access only `damfileA`:

If the accessed blocks do not overlap,  $10 \times 5 = 50$  cache blocks are

connected to the cache block chain for `damfileA`. This is because data is stacked in the cache memory to prevent another process from referencing the latest block data and the updated data before the transaction determination while the transaction is being executed.

- When only one transaction branch accesses `damfileA`:

If the accessed blocks do not overlap, data is stacked in the cache memory as in the above case and 10 cache blocks are connected to the cache block chain for `damfileA`.

As described above, when you specify 0 as the threshold for the number of cache blocks, the number of cache blocks connected to the cache block chain depends on the type of access made by the UAP and the values set in the DAM service definition. The same phenomenon occurs when you specify a small value as the threshold for the number of cache blocks. A small value refers to a number smaller than the number of blocks that are accessed in a transaction executed by a UAP.

In Example 3, part of the DAM service-exclusive shared memory is not used as in Example 2.

### (c) Settings according to the type of access made by UAPs

Imagine that the following information is output after you execute the `damchinf` command:

```
CleanUP Count:1  Next CleanUP FILE-No:1  Using Rate:80%
FileNo FileName BlkLen BlkNum CchBlkNum PreservNum LimitNum
ReUse
  1 damfile1    504 10000    7900      0      -1 Exist
  0 damfile0    504 10000     100      0      -1 Exist
  2 damfile2    504 10000      0      0      -1 None
```

This section explains the suitable threshold for the number of cache blocks for each type of access made by a UAP.

- If the UAP hardly accesses `damfile1` any more:

Since `Using Rate:80%` is output, only 20% of the cache block area can be allocated to `damfile0` and `damfile2`. From now on the UAP will hardly access `damfile1` at all, so the cache block area for 7,900 cache blocks that are connected to the cache block chain for `damfile1` remains secured. You can specify a small value as the threshold for `damfile1` to decrease the secured cache block area and increase the cache block area to be allocated to `damfile0` and `damfile2`.

Since many cache blocks are connected to the cache block chain for `damfile1`, clean-up processing will be performed if the cache block area becomes

insufficient. Since `Next CleanUP FILE-No:1` is output, you can tell that the cache block chain for `damfile1` is the target of clean-up. When 7,900 cache blocks are released, the performance degrades rapidly. To prevent this degradation, you should specify a threshold for `damfile1`.

- When the UAP accesses `damfile1` frequently:

When the UAP accesses `damfile1` frequently, the cache block chain for `damfile1` is frequently searched. By specifying a threshold, you can shorten the cache block chain, reducing the chain search time. However, if the threshold is too small, the cache efficiency decreases, degrading the performance. Determine an optimum value by testing several thresholds.

- When the UAP accesses all the DAM files uniformly:

When the UAP accesses all the DAM files uniformly, the cache block area allocated to each DAM file should be uniform. By setting the same value as the threshold for each DAM file, the cache block area used by each DAM file becomes uniform.

#### (d) Notes

Note the following:

- If you do not specify a threshold for the number of cache blocks, cache blocks will be secured until the DAM service-exclusive shared memory becomes full. If you specify a threshold for the number of cache blocks, cache block securing will be suppressed in the middle, causing some part of the shared memory area to be unused.
- The number of cache blocks connected to the cache block chain for each DAM file may exceed the value set for the threshold.
- If you specify an unspecifiable threshold in the `damch1mt` definition command in the DAM service definition, the part with the invalid specification is shown in the KFCA00219-E message. The KFCA01644-I message indicates that the default is used. In this case, the threshold managed by the corresponding DAM file is not set and cache blocks are secured until the shared memory resource is used up.

#### (13) Specifying the search sequence when reusing cache blocks

In the `dam_cache_reuse_from` operand in the DAM service definition, you can specify the first cache block you want to reuse in the cache block chain when a transaction that accesses a DAM file requires a new cache block.

When you specify `last`, cache blocks are reused from the oldest cache block connected to the cache block chain.

When you specify `first`, cache blocks are reused from the latest cache block connected to the cache block chain.

The default is `last`.

#### **(14) Specifying the boundary for reusing cache blocks**

In the `damcache` definition command in the DAM service definition, you can specify the boundary for reusing cache blocks. If you have specified a boundary, when securing new cache blocks, the reuse of cache blocks for another DAM file takes priority over reuse of cache blocks for the accessed DAM file. This applies if the number of cache blocks has not reached the boundary for reusing cache blocks.

If the number of cache blocks exceeds the boundary for reusing cache blocks, the cache blocks for the accessed DAM file are reused. Therefore, when you specify 0 for the boundary for reusing cache blocks, cache blocks for the accessed DAM file are reused as in the usual processing.

In the `dam_default_cache_num` operand in the DAM service definition, you can specify the boundary for reusing cache blocks in a logical file without the `damcache` definition command specified.

Specifying a boundary is effective when you want to heavily access many blocks in a DAM file and then frequently access another DAM file, which means that a small number of DAM files occupy cache blocks and only a few cache blocks are allocated to a different DAM file that is accessed frequently.

When you access multiple DAM files at random, which means that cache blocks are allocated to each DAM file uniformly, specifying a small value for the boundary for reusing cache blocks releases cache blocks frequently, increasing memory securing and data read processing. This may degrade the performance.

#### **(a) Example that improves the performance**

For the purpose of explanation, two DAM files are used here and the block lengths are all the same. The maximum number of cache blocks that can be secured in the entire system is 100. A UAP references 98 blocks of DAM file A and then starts to access DAM file B.

When the UAP accesses DAM file B, the number of blocks it references increases by one in each access. The UAP will reference up to 50 blocks.

1st access: Blocks 1 and 2 are referenced.

2nd access: Blocks 1, 2, and 3 are referenced.

3rd access: Blocks 1, 2, 3, and 4 are referenced.

:

49th access: Blocks 1, 2, 3, ... and 50 are referenced.

Under these conditions, the difference between specifying and not specifying the `damcache` definition command is described below.

When you do not specify the `damcache` definition command:

1. When the UAP accesses DAM file B for the first time, the DAM service secures the cache block area from the shared memory. At this time, 98 cache blocks are already secured for DAM file A. Therefore, the DAM service can secure only two cache blocks. The DAM service reads the data of blocks 1 and 2 into the cache blocks and connects the cache blocks to the cache block chain for DAM file B.
2. When the UAP references blocks 1, 2, and 3, file I/O operations are not performed for blocks 1 and 2 since their data exists in the cache. The UAP accesses block 3 for the first time. Therefore, the DAM service attempts to secure a cache block for block 3 to read data. However, since free space runs out in the shared memory in step 1, cache blocks for DAM file B are reused. For reuse processing, the DAM service disconnects the cache block containing the data of block 1 from the cache block chain, reads the data of block 3 into the cache block, and then connects the cache block to the beginning of the cache block chain.
3. When the UAP references blocks 1, 2, 3, and 4, cache blocks are reused as in step 2. The DAM service releases the cache block for block 2, reads block 1 into the cache block, and connects the cache block to the beginning of the cache block chain. Then to read the data of block 2, the DAM service reuses the cache block of block 3 and reads block 2. This processing is repeated for blocks 3 and 4.
4. When step 3 is repeated, the DAM service reuses cache blocks for DAM file B, which means that the DAM service frequently releases cache blocks from the cache block chain and frequently executes file I/O operations to read data. About 1275 release operations are performed and about 1,275 file I/O operations are performed.

When you specify the `damcache` definition command:

You specify 50 as the boundary for reusing cache blocks for DAM file A and specify 90 as the boundary for reusing cache blocks for DAM file B.

1. The DAM service secures two unused cache blocks as in step 1 when you do not specify the `damcache` definition command, reads the data of blocks 1 and 2 into the cache blocks, and connects the cache blocks to the cache block chain.
2. When the UAP references blocks 1, 2, and 3, file I/O operations are not performed for blocks 1 and 2 since their data exists in the cache. For the cache block for reading the data of block 3, the DAM service releases and secures a cache block for a DAM file whose number of cache blocks exceeded the boundary for reusing cache blocks (DAM file A in this case). This release and securing of a cache block is performed because since the number of cache blocks for DAM file B does not exceed the boundary for reusing cache blocks. The DAM service reads the data of block 3 into the secured cache block and connects the cache block to the cache block chain. Three cache blocks are now connected to the cache block chain for DAM file B.



3. When the UAP references blocks 1, 2, 3, and 4, file I/O operations are not performed for blocks 1, 2, and 3 since their data exists in the cache. When the DAM service reads the data of block 4, a single file I/O operation is executed to reuse the unused cache block for DAM file A.
4. Step 3 is repeated until the number of cache blocks for DAM file A exceeds the boundary for reusing cache blocks. The release from the cache block chain is executed approximately 50 times and file I/O is also executed about 50 times, improving the performance.

**(b) Example that degrades the performance**

The conditions, such as the block lengths in DAM files, are the same as the previous example. DAM files are accessed as follows:

1st access: Blocks 1 to 50 of DAM file A are referenced.

2nd access: Blocks 1 to 50 of DAM file B are referenced.

3rd access: Blocks 51 to 100 of DAM file A are referenced.

4th access: Blocks 1 to 50 of DAM file B are referenced.

:

The UAP references DAM files A and B alternately. The UAP references the next 50 blocks of DAM file A for each access.

Since the maximum number of cache blocks is 100, 50 cache blocks of DAM file A and 50 blocks of DAM file B are connected by the second access. For the third and later accesses, the processing differs depending on whether you specify the `damcache` definition command.

- When you do not specify the `damcache` definition command:

During the third access, the DAM service releases the cache blocks for blocks 1 to 50, replaces the old data with the data of blocks 51 to 100, and connects the cache blocks to the cache block chain.

For the fourth access, all the blocks are found in the cache. Therefore, the DAM service does not release cache blocks and file I/O operations are not performed.

During the later access, for DAM file A, the DAM service releases all the cache blocks and file I/O operations are performed like the third access. However, for DAM file B, the DAM service does not release cache blocks and file I/O operations are not performed. When the number of blocks that can be referenced in DAM file A is 1,000, cache blocks are released 950 times and file I/O is performed 1,050 times for both DAM files A and B.

- When you specify the `damcache` definition command:

You specify 100 as the boundary for reusing cache blocks for DAM file A and

specify 50 as the boundary for reusing cache blocks for DAM file B.

During the third access, when the UAP references blocks 51 to 100 in DAM file A, 50 cache blocks need to be secured. At this time, since the number of cache blocks for DAM file A does not exceed the boundary for reusing cache blocks, the DAM service uses the cache blocks for a different DAM file. The DAM service forcibly reuses the cache blocks for DAM file B since there is no free space in the shared memory even if the number of cache blocks for DAM file B does not exceed the boundary for reusing cache blocks. Therefore, the DAM service releases the cache blocks containing the data of blocks 1 to 50 in DAM file B, replaces the old data with the data of blocks 51 to 100 of DAM file A, and connects the cache blocks to the beginning of the cache block chain for DAM file A. At the end of the third access, the cache blocks for blocks 1 to 100 are connected to the cache block chain for DAM file A and there is no cache block for DAM file B.

For the fourth access, when the UAP references blocks 1 to 50 in DAM file B, 50 cache blocks need to be secured. Like the third access, the DAM service forcibly reuses the cache blocks for DAM file A, releases the cache blocks containing the data of blocks 1 to 50 of DAM file A, stores the data of blocks 1 to 50 of DAM file B, and connects the cache blocks to the cache block chain for DAM file B.

The same processing is repeated for the third and fourth access. When the number of blocks that can be referenced in DAM file A is 1,000, cache blocks are released 1,900 times and file I/O is performed 2,000 times for both DAM files A and B, degrading the performance compared to not specifying the `damcache` definition command.

### (c) Notes

Note the following:

- When the number of cache blocks for the accessed DAM file does not exceed the boundary for reusing cache blocks, cache blocks for a different DAM file are reused. If all the cache blocks for the target DAM file are being accessed, cache blocks for the accessed DAM file are reused. Therefore, cache blocks may be reused even if the boundary for reusing cache blocks is not exceeded.
- If you execute the `damadd` command without specifying the `-l` option, the boundary for reusing cache blocks for that particular DAM file is 0 and the boundary does not depend on the value specified in the `dam_default_cache_num` operand in the DAM service definition.
- When you specify an unspecifiable value for each boundary for reusing cache blocks in the DAM service definition, the following occurs:
  - When you specify an unspecifiable value for the `dam_default_cache_num` operand:

The system outputs the KFCA00216-E message to indicate the invalid specification. The system also outputs the KFCA01644-I message to show that the default is used. In this case, 0 is used as the boundary for reusing cache blocks for the DAM file for which the boundary is not specified using the `damcache` definition command.

- When you specify an unspecifiable value for the `damcache` definition command:

The system outputs the KFCA02529-E message to indicate the invalid specification. The system also outputs the KFCA01644-I message to show that the default is used. In this case, the value specified for the `dam_default_cache_num` operand is used as the boundary for reusing cache blocks for the applicable DAM file. If the `dam_default_cache_num` operand is not specified or if an unspecifiable value is specified, 0 is assumed.

#### **(15) Cacheless access for unrecoverable DAM files**

As described in 4.4.1(11) *Using cache blocks*, the DAM service stores the block data in the DAM files that are once read in the DAM service-exclusive shared memory to reduce the number of file I/O operations.

However, for the unrecoverable DAM files only, you can directly perform file I/O operations and return the block data in the disk to UAPs. This is called *cacheless access*. To enable cacheless access for the unrecoverable DAM files, specify the `-f` option of the `damfile` definition command in the DAM service definition.

When you specify cacheless access, file I/O operations are performed for each reference and update, and performance may degrade.

When you use DAM files with cacheless access specified, you can reduce the amount of memory required by obtaining the appropriate size of DAM service-exclusive shared memory and then specifying that value in the `dam_cache_size_fix` operand in the DAM service definition. For details, see the description of the `dam_cache_size_fix` operand in the DAM service definition in the manual *OpenTP1 System Definition*.

#### **4.4.2 User files: TAM files (TP1/FS/Table Access)**

The TAM file service provides faster access to user data because it accesses user data in memory. The user data is loaded from a user file. TAM files are managed by part of OpenTP1 called the *TAM file service*.

The TAM file service uses:

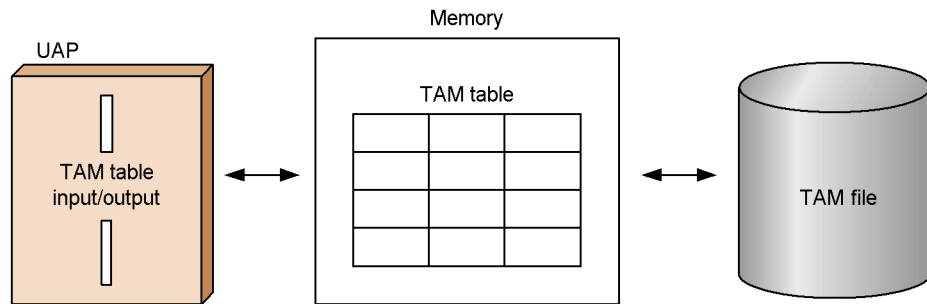
- TAM files
- TAM tables

A *TAM file* is made by the OpenTP1 administrator with the command `tamcre`. A TAM file consists of pairs of values: a data field to contain the *record*, and another field to contain an associated *key value*. When the TAM file is created, the actual records and key values can be input automatically from another file.

The TAM file is loaded into memory as a *TAM table*, which is a simple structure table. To load the table, the OpenTP1 administrator defines the TAM service definition, which associates TAM tables and TAM files and specifies a time when the TAM file is to be loaded. At the specified time, the TAM file is loaded into memory as a TAM table.

TAM tables are in a one-to-one correspondence with TAM files, as shown in Figure 4-18.

Figure 4-18: TAM tables and TAM files



**(1) Comparing TAM and DAM files**

Like DAM files, TAM files store user data in direct-organization files. The table access method, however, provides high-speed access to TAM files. A drawback is that such tables require more memory than required for DAM files.

When memory is abundant, you can configure a high-performance OpenTP1 file system using more TAM files. When memory is scarce, use DAM files. The same functions used for DAM file I/O can be used for TAM files also. So when memory is added, user data can be moved from DAM files to TAM files without changing the UAPs that have been used. Using the same functions also enables moving part of the user data to TAM files and leaving the rest of the user data in DAM files. For details about tree format and hash format, see 4.4.2(2) *Internal functioning of a TAM table*.

Table 4-10 lists some special features of TAM tables and files.

Table 4-10: Special features of TAM tables and files

Item	Feature
File structure	Direct organization file (an OpenTP1 file).

Item	Feature
Record format	Fixed length, non-blocked format.
File access	A block of data can be referenced and updated by the table access methods using key values. There are three types of files: reference-only files, updatable files that prohibit addition and deletion, and updatable files that allow addition and deletion.
Relation to transaction	A record tentatively updated during transaction processing is actually updated in memory at the commit operation on a transaction. If the transaction is rolled back, the tentatively updated contents of the record are discarded. The record actually updated in memory is passed to a TAM file. OpenTP1 regularly writes only the records that were actually updated in memory to TAM files.
File backup and restore	The TAM file management command <code>tambkup</code> can backup a TAM file to another file, and the command <code>tamrstr</code> can restore the file. Also, the most recent contents of a TAM file can be recovered by using the backup file and the recovery journals after the backup was obtained. Recovery of a TAM file uses the TAM FRC facility.
Table addition	The TAM file management command <code>tamadd</code> can add a TAM table into an online system during OpenTP1 operation. The physical file for the TAM table must be made before executing <code>tamadd</code> . An added TAM table has an initial shutdown status. Use the command <code>tamrles</code> to release the shutdown.
File deletion	The TAM file management command <code>tamhold</code> can logically shut down a TAM table during system operation. You can then use the command <code>tamrm</code> to take the shutdown TAM table off the system without stopping the system. You can use the command <code>tamdel</code> to delete a TAM file, corresponding to a TAM table, from OpenTP1.

If plenty of memory is available, you can configure a high-performance system using TAM files. If memory is limited, use DAM files instead.

The functions used for DAM file I/O can also access TAM tables. So when more memory is added, you can move user data from the DAM files to TAM files without needing to change your existing UAPs. Alternatively, you can move some user data to TAM files, and leave some in DAM files.

## (2) Internal functioning of a TAM table

This subsection describes the internal working of *tree-format* and *hash-format* TAM tables in OpenTP1. This subsection is not necessary for understanding the benefits of TAM tables, and readers can skip this material unless they have a particular interest in it.

In a tree-format TAM file, records are managed in a tree structure according to whether the value of each record is larger or smaller than the intermediate key value of the tree root. Because records are located and accessed by comparing the key values in the tree one-by-one, record retrieval takes longer than with hash-format TAM files.

In a hash-format TAM file, OpenTP1 locates and accesses a record via a number called the *hash value*. OpenTP1 uses an internal *hash function* to generate a *hash value* (a random number) from a key value.

The manner in which a key value is specified in a UAP when accessing a TAM table differs for tree format and hash format.

Table 4-11 shows the differences.

Table 4-11: Processing when a UAP specifies a key value for TAM table access

Specification in UAP	TAM table made from tree-format TAM file	TAM table made from hash-format TAM file
To retrieve one record: key-value = n	The record with the specified key value is retrieved. If such a record is not found, an error is returned.	Same as tree-format
To retrieve more than one record in a TAM table made from a tree-format TAM file: key-value < n key-value > n key-value ≤ n key-value ≥ n	The first record with the nearest key value to the specified key value is retrieved.	An error is returned.
To retrieve the first record in a TAM table made from a hash-format TAM file <sup>#</sup>	An error is returned.	The first record of the TAM table is retrieved ignoring the key value.
To retrieve the next (NEXT specification) record in a TAM table made from a hash-format TAM table <sup>#</sup>	An error is returned.	The record next to the one with the specified key value is retrieved.

#

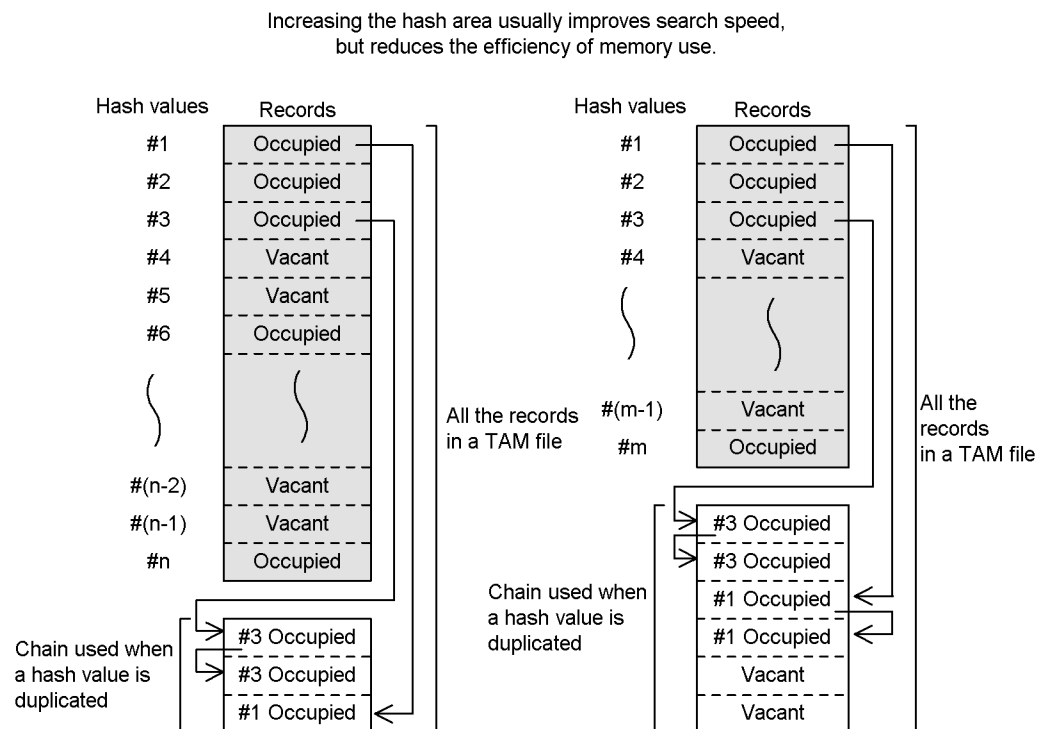
For a TAM table made from a hash file, all the records in a TAM table can be searched by the combination of the first record retrieval and the NEXT retrieval; however, this all-record search can be performed only when the records contain key values. If the key values are removed from the records, the user cannot know the key value to be specified for the NEXT retrieval. For further details, see the manual *OpenTP1 Operation*.

For a hash-format TAM table, you can specify how much of the total record space to use as the hash area as an argument of the `tamcre` command when creating the TAM file. With a larger hash area, record retrieval is faster because there is less duplication of hash values and the list needs to be searched less often. However, when the list area decreases and the list gets full, any further duplication of hash values will prevent further records from being added even if the hash area has free space, resulting in poor

memory efficiency. With a smaller hash area, duplication of hash values increases and the list has to be searched more often. This makes record retrieval slower, although memory usage is more efficient.

Figure 4-19 compares TAM tables that use large or small hash areas.

Figure 4-19: TAM tables that use large and small hash areas



### (3) Creating a TAM file

To create a TAM file:

1. Create the OpenTP1 file system.
2. Use the TAM file management command `tamcre` to create the new TAM file.

If you want to store initial data in a TAM file at the time the TAM file is created, first store the initial data in a file, then specify the filename in the argument of `tamcre`. OpenTP1 will copy the initial data from the file to the TAM file. The file to store the initial data of a TAM file is called a *TAM data file*.

### (4) Access modes for a TAM table

An *access mode* is the TAM table attribute that restricts UAP access to a record. By setting the appropriate access mode, you can differentiate use of TAM files according

to their purposes, and enable record management of TAM tables.

In the TAM service definition, you can specify the access mode for a TAM table that is to be installed in an online system at the same time that OpenTP1 starts. For a TAM table that is to be added online during OpenTP1 operation, you can specify the access mode in the `-a` option of the TAM file management command `tamadd` that adds the TAM table.

Depending on how a UAP accesses the TAM table records, TAM tables can be classified into three types. A *reference-only* TAM table allows records to be referenced only: no record can be added, deleted, or updated during online processing. A TAM table that is *updatable but prohibits addition and deletion* allows updating of records during online processing but prohibits adding and deleting of records. A TAM table that is *updatable but permits addition and deletion* permits adding, deleting, and updating records during online processing.

### (5) Loading and unloading a TAM table

OpenTP1 loads the TAM table stored in the TAM file into memory at a user-specified time called the *loading opportunity*. For a TAM table that is to be installed in the online system when OpenTP1 starts, you must define the loading opportunity in the TAM service definition. For a TAM table that is to be added to an online system during OpenTP1 operation, you must specify the loading opportunity in a parameter of the command `tamadd` that adds the TAM table.

A TAM table loaded by the command `tamload` is unloaded by the command `tamunload`. The TAM file loaded when OpenTP1 started or when it was added to the online system is automatically unloaded at the termination of OpenTP1. Also, a TAM file loaded by a function (`dc_tam_read()`) issued from a UAP is automatically unloaded when the function `dc_tam_close()` is issued to close the TAM file. Table 4-12 shows the loading opportunities and unloading methods for TAM tables.

Table 4-12: Loading opportunities and unloading methods for TAM files

Time TAM table is installed in an online system	Loading opportunity (one of the three opportunities is selected)	Unloading method <sup>#3</sup>
At the start of OpenTP1 <sup>#1</sup>	When the TAM file service starts with OpenTP1	Automatically at TAM termination
	When the <code>tamload</code> command to load the table is entered	By command
At the start of OpenTP1 <sup>#1</sup>	When the TAM table is opened by the <code>dc_tam_open()</code> function issued from a UAP	Automatically at the close of table
During OpenTP1 operation <sup>#2</sup>	When the TAM table is cataloged in the online system	Automatically at TAM termination



Time TAM table is installed in an online system	Loading opportunity (one of the three opportunities is selected)	Unloading method <sup>#3</sup>
	When the <code>tamload</code> command to load the table is entered	By command
	When the TAM table is opened by the <code>dc_tam_open()</code> function issued from a UAP	Automatically at the close of table

#1

The loading opportunity is specified in the TAM service definition.

#2

The loading opportunity is specified in an option of the command `tamadd`, which adds a TAM table.

#3

The TAM table is unloaded at the termination of the TAM file service regardless of the loading opportunity.

### (6) Locks on a TAM table

OpenTP1 can lock TAM tables with *table-based locks* or *record-based locks*. When a TAM table or record has already been locked by another UAP, you can use a parameter of a function (`dc_tam_open()` or `dc_tam_read()`) to specify whether or not to wait until the lock is released. In the lock service definition, you can specify how long to wait for the lock release; this time is called the *lock monitoring time*. If the lock is not released before the lock monitoring time expires, an error is returned. For TAM tables, a lock takes effect only within each global transaction.

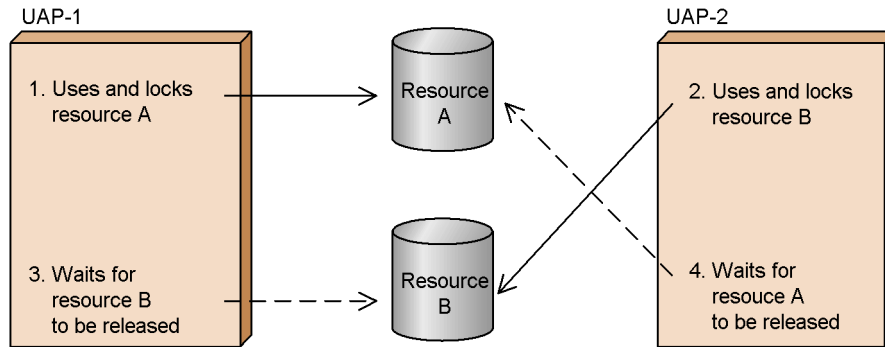
When more than one UAP can access both a TAM table and a DAM file, a *deadlock* might occur among the UAPs. If a deadlock occurs, OpenTP1 uses the UAP priority specified in the user service definition to recover from the deadlock. See 3.9.1(5) *Deadlocks in TAM and DAM files* for details about deadlocks.

### (7) Deadlocks in TAM and DAM files

This section describes how TAM and DAM lock functions are affected by deadlocks.

When two or more UAPs lock multiple resources in different sequences, an undesirable situation called a *deadlock* might occur. For example, in one form of deadlock, one UAP might lock resource A and wait for resource B to be released while another UAP might be locking resource B and waiting for resource A to be released. Figure 4-20 illustrates a deadlock.

Figure 4-20: Example of a deadlock



OpenTP1 detects deadlocks differently depending on whether the UAPs are on the same or different nodes. When UAPs are running on the same node, OpenTP1 automatically checks at regular intervals to detect deadlocks. When UAPs are running on different nodes, however, OpenTP1 cannot directly detect a deadlock so OpenTP1 checks whether any UAP has exceeded the specified time-limit for waiting for a resource. In the `lck_wait_timeout` operand in the lock service definition, you can specify the time-limit for waiting for a resource.

### (8) Online backup for a TAM file

You can back up a TAM file during an OpenTP1 job. This is an *online backup*. Execute the `tambkup` command with the `-o` option to back up a TAM file during online processing. Without the `-o` option specified, a TAM file is backed up offline. This is how you back up offline:

1. Execute the `tamhold` command.  
The logical file is logically shut down.
2. Execute the `tamrm` command.  
The logically shut down logical file is separated from online processing.
3. Execute the `tambkup` command without the `-o` option.  
The TAM file is backed up.
1. Recovering a TAM file using files that were backed up online reduces the number of unload journal files to be used for recovery. This reduces the time required for recovering a TAM file compared to the time it takes using files that were backed up offline.

### (9) Backing up and restoring TAM files using standard input/output

You can use standard output to the backup destination of a TAM file and standard input

to the input file to be restored. Using standard input/output can redirect a command. If you use standard input/output, specify the `-s` option in the `tambkup` and `tamrstr` commands, then execute them.

#### 4.4.3 IST service (TP1/Shared Table Access)

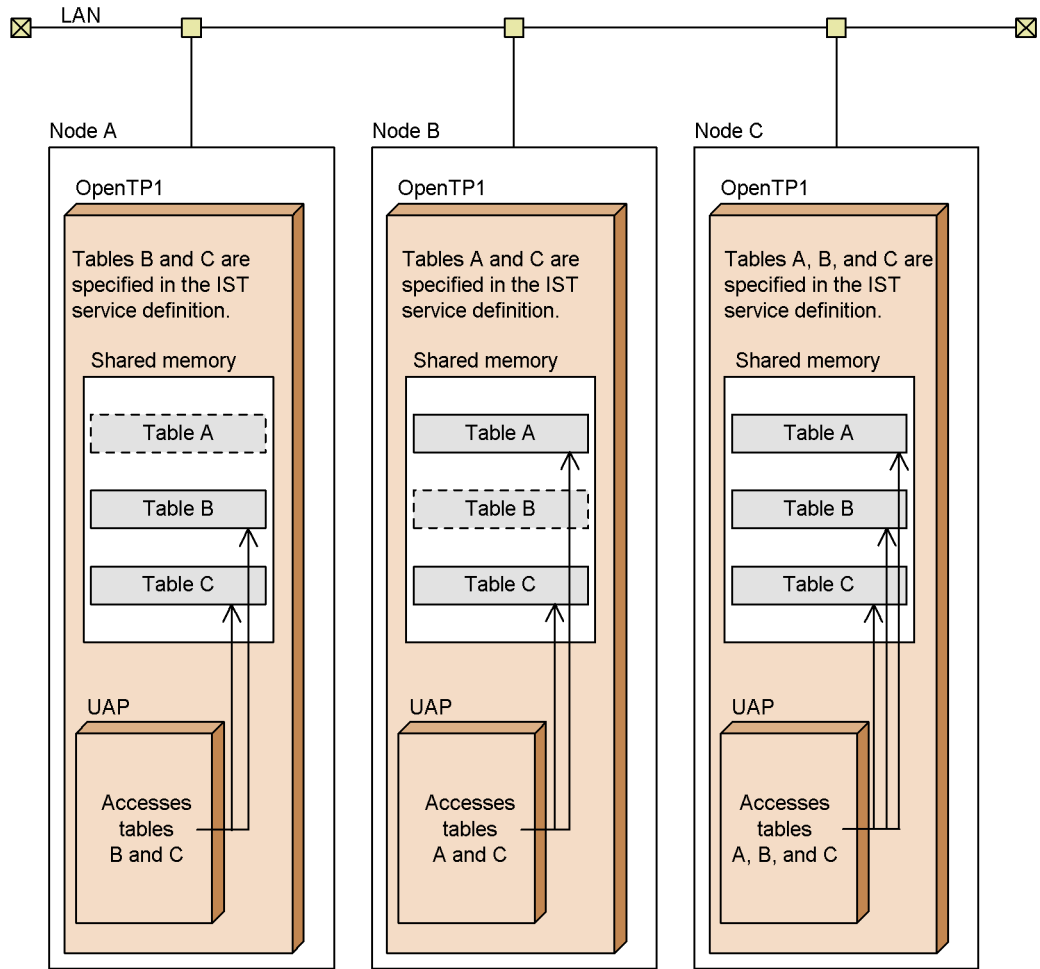
The IST service allows information from all nodes in a group to be shared among the nodes and to be accessed from any node in the group. The information is stored in special tables called *internode shared tables* (ISTs). The IST service stores internode shared tables in shared memory, and enables UAPs to reference and update the tables without knowing the actual physical locations of the information that makes up the tables.

To use internode shared tables, *TP1/Shared Table Access* must be installed on each node.

As an example of how internode shared tables can be used, you could place job-status information from various nodes into internode shared tables. This would allow you to check the job status of all nodes from any node. This shared access to job-status information simplifies the management of jobs.

Figure 4-21 illustrates the configuration of the IST service.

Figure 4-21: IST service configuration



Legend:

- ▭ : Accessible IST table
- ▭ (dashed border) : Inaccessible IST table

To use the IST service, make sure that all the nodes have the same IST table definition.

In Figure 4-21, different table names are specified in the IST table definitions of nodes A, B, and C. Thus, nodes A and B continue to output the KFCA25533-W message periodically until OpenTP1 terminates. This message notifies you that invalid table information was received.

The IST service is not recommended, however, for the following cases when distributing data to multiple nodes:

- Processing that requires the immediate distribution of data
- Processing that handles large quantities of data
- Processing that updates data frequently

**(1) Access environment for internode shared tables**

Internode shared tables exist in memory shared by each node. No actual file, such as TAM, DAM, or ISAM, stores an internode shared table. UAPs can access internode shared tables during online processing only; the tables cannot be accessed offline.

**(2) Structure of an internode shared table**

An internode shared table consists of multiple records. When a UAP references or updates an internode shared table, the table data is accessed in units of records, and one request from a UAP can access one record or access multiple records.

**(3) Accessing an internode shared table**

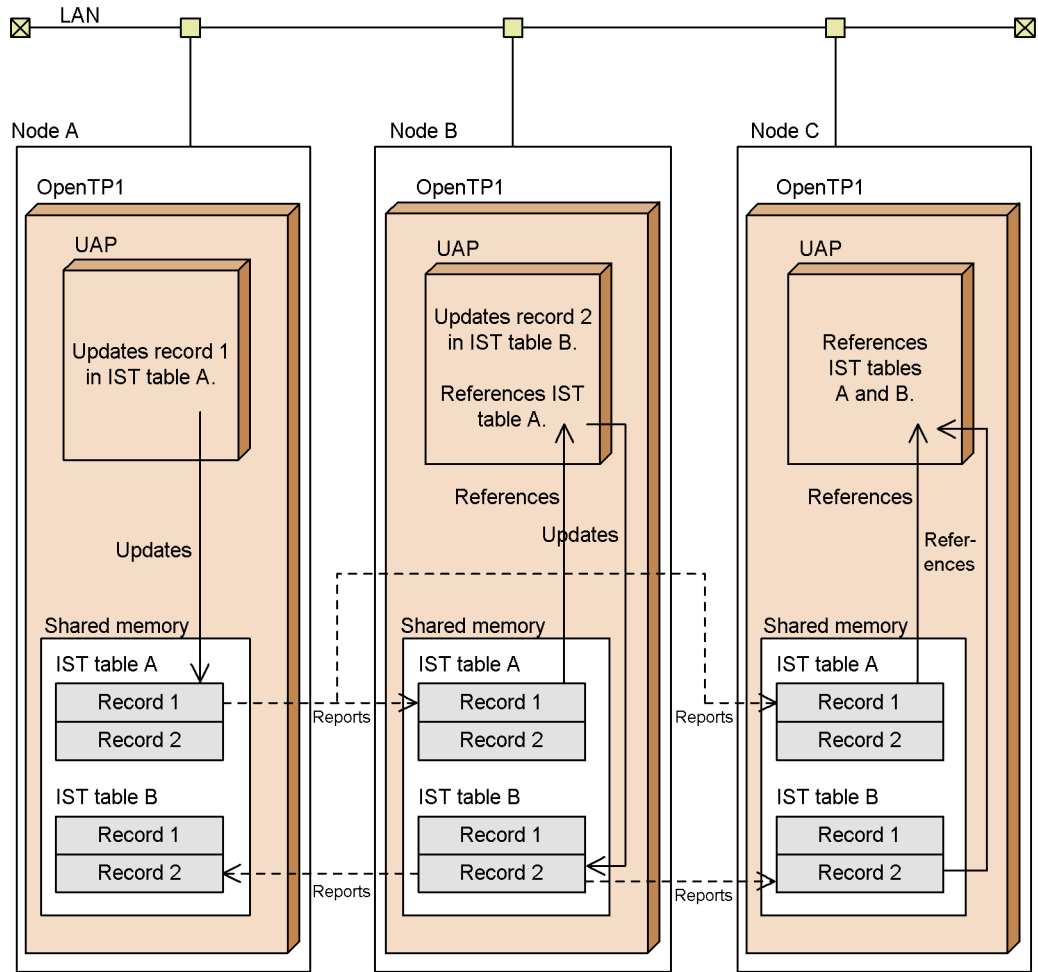
For details on how to access an internode shared table from a UAP, see the *OpenTPI Programming Guide*. Transaction functions cannot be used for commit or rollback operations when accessing an internode shared table.

An internode shared table is locked for each function called from the UAP. Access to an internode shared table is not monopolized during the period from input to changing of data, so no deadlock will occur even if multiple UAPs access one table.

**(4) Example of using the IST service**

Figure 4-22 shows an example of effective use of the IST service.

Figure 4-22: Example of effective use of the IST service



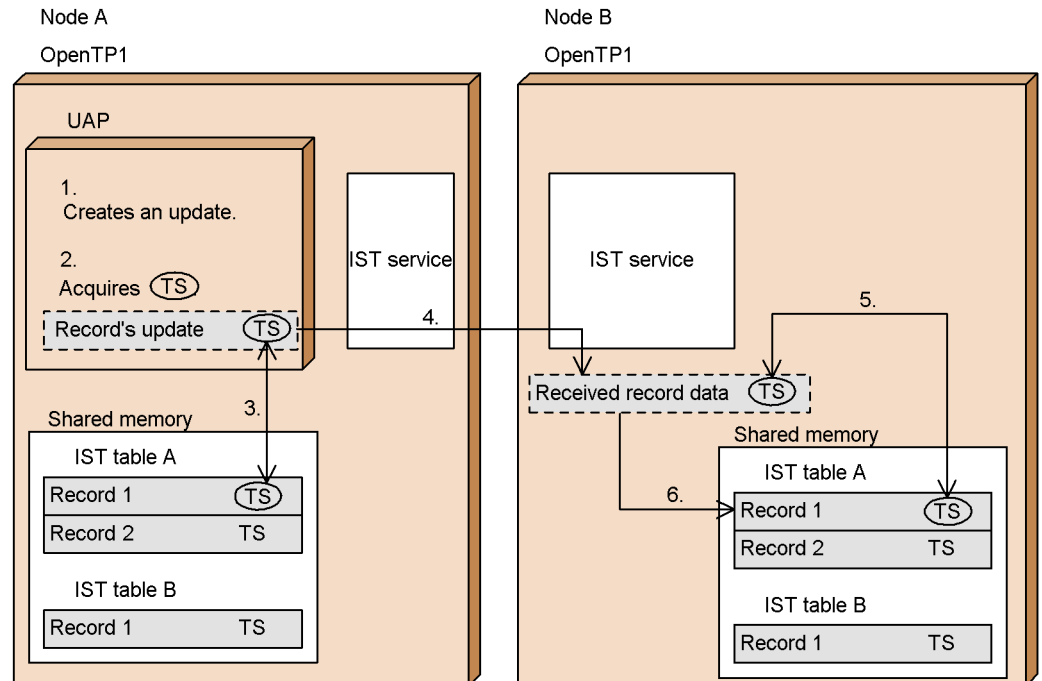
In Figure 4-22, only node A updates IST table A, and only node B updates IST table B. When data is updated at any node, the IST service reports and applies the change to other nodes. Thus, any node can reference up-to-date IST tables. This configuration is effective in that only one specific node can update a specific IST table.

For example, suppose that node A writes its status information to IST table A, and node B writes its status information to IST table B. In this case, if you create IST table C in which node C can write and update its status information, each node can reference the other nodes' status information. Note that there is a delay due to communication with other nodes for applying the change to their IST tables. Before the change is applied, other nodes reference the old information.

**(5) Notes about using the IST service**

When multiple nodes use the IST service, make sure that there is no time difference among the nodes. If there is a time difference, updates at a node might not be correctly applied to other nodes. The following figure illustrates how the IST service updates IST records (records in the IST tables) on multiple nodes.

Figure 4-23: How IST records are updated



Legend:  
TS: Timestamp

1. A user application program (UAP) creates an update of IST record 1, in IST table A on node A.
2. The UAP acquires the current machine time at microsecond accuracy, and assigns the time to the record's update as a timestamp.
3. OpenTP1 compares the assigned timestamp with the timestamp of the corresponding IST record in the shared memory on node A.

If the timestamp of the record's update is newer, OpenTP1 updates the IST record in the shared memory. If the timestamp of the record's update is older, OpenTP1 does not update the IST record in the shared memory. The `dc_ist_write` function returns normally even when it does not update any IST records.

4. If an IST record in the shared memory is updated, the IST service on node A notifies the IST service on node B that an IST record was updated on node A. At this time, the updated IST record and its timestamp are also reported.
5. Upon reception of the updated IST record and its timestamp, the IST service on node B compares the timestamp of the received IST record with the timestamp of the corresponding IST record on node B.
6. If the timestamp of the received IST record is newer, the corresponding IST record on node B is replaced with the received one.

As described above, the IST service uses the timestamp to decide whether to update the IST record. Note, however, the IST service may be unable to apply the latest data in the following cases:

- When node A's machine time is ahead of node B's

If an IST record is updated on node A and then the same IST record is updated on node B, the timestamp assigned on node A may be newer. In this case, if the timestamp assigned on node A is newer, the record updated on node B is not applied to node A.

Also, when the IST record updated on node A is reported to node B, the IST service assumes that the reported IST record is the latest one. In this case, the reported IST record replaces the one that was actually updated later.

- When node A's machine time is behind node B's

- When an IST record was updated on node B and the updated IST record has already been reported to node A

After the IST record update on node B, even if the same IST record is updated on node A, the `dc_ist_write` function returns normally without updating the record.

- When an IST record was updated on node B but the updated IST record has not yet been reported to node A

After an IST record is updated on node B, if the same IST record is updated on node A, the IST record updated on node A is at first applied to node B. However, because the IST service assumes that the timestamp of the IST record reported by node B is newer, the IST service then applies the older reported update to node A.

#### 4.4.4 User files: ISAM files (ISAM and ISAM/B)

ISAM (indexed sequential access method) files are one of the three types of user files used in OpenTP1 application processing. For details about the ISAM file service, see the manual *Indexed Sequential Access Method ISAM*.



**(1) Overview of ISAM files**

An indexed sequential access file consists of an index part for key management and a data file part for data storage. The use of keys allows sequential access and random access processing.

An ISAM file can be manipulated by calling a library function from a UAP or by executing a utility command for ISAM file management.

**(2) Types of ISAM services**

The following ISAM file services can be accessed from an OpenTP1 UAP.

**(a) ISAM service**

ISAM files are handled as regular files. Processing is not synchronized with OpenTP1 transaction processing.

**(b) ISAM/B service**

ISAM files are handled in synchronization with OpenTP1 transaction processing. Use of the ISAM/B service ensures file integrity when a transaction is committed or rolled back.

- Prerequisite products for ISAM/B

To use the ISAM/B service, the ISAM transaction facility (ISAM/B) must be available in addition to ISAM.

- Area for file creation

ISAM files handled by ISAM/B are created in the area allocated to the OpenTP1 file system.

- Difference from OpenTP1 file service (TP1/FS/xxx)

ISAM/B does not use the lock service. Therefore, if a deadlock occurs, the OpenTP1 lock service facility (lock scope reduction based on priority and deadlock information output) is not available.

**4.4.5 Accessing database management systems**

This section describes how database management systems (DBMSs) can be used in OpenTP1 UAP.

**(1) Relation to OpenTP1 transaction processing**

The usage of DBMSs depends on whether the DBMS supports the XA interface in the X/Open DTP model, and whether the DBMS can work with OpenTP1 transactions.

**(a) DBMSs that support the XA interface**

Only DBMSs that support the XA interface, for example ORACLE, can be controlled by OpenTP1 transaction processing. When a DBMS supports the XA interface,

updates are possible using the commit and rollback operations of OpenTP1 transaction processing. In such transaction processing, you can use the functions that control OpenTP1 synchronization points (such as the functions `dc_trn_begin()`, `dc_trn_unchained_commit()`, `tx_begin()`, or `tx_commit()`).

Facilities provided by a DBMS for controlling transactions cannot be used.

In UAPs that access multiple databases, OpenTP1 allows updates while protecting the consistency of the multiple databases. The OpenTP1 resource managers (provided by the products TP1/FS/Direct Access, TP1/FS/Table Access, TP1/Message Control, TP1/Message Queue, and Hitachi ISAM and ISAM/B) support the XA interface. Thus, a UAP can process OpenTP1 transactions when accessing DBMSs that conform to the XA interface in the same way it does when it accesses the OpenTP1 resource manager. Even when some failure cause an abnormal termination of a UAP or when OpenTP1 is restarted, OpenTP1 performs a transaction determination (i.e., decides whether to perform a commit or rollback) for both the DBMS and the OpenTP1 resource manager.

**(b) DBMSs that do not support the XA interface, or DBMSs that do not work with OpenTP1 via the XA interface**

A DBMS that does not support the XA interface can be accessed, but cannot be synchronized with OpenTP1 transactions.

When a DBMS does not work with OpenTP1 via the XA interface, OpenTP1 cannot order a transaction determination to the DBMS in certain situations: such as when a UAP abnormally terminates during access to a database, or when OpenTP1 requires a complete-recovery restart during access to a database. In such situations, you must recover the transactions using the DBMS facilities.

**(2) Preparing to use XA-compliant databases**

The following tasks are required when DBMSs that support the XA interface cooperate with OpenTP1 via the XA interface:

- registering the DBMS in OpenTP1
- linking
- writing system definitions
- setting environment variables

**(a) Registering the DBMS in OpenTP1**

Register several names for resource managers other than those provided by OpenTP1. Use either of these methods to register them in OpenTP1:

- After setting up OpenTP1 using the `dcsetup` command, execute the `trnlkrm` command.
- Create an extended RM registration definition.

When an extended RM registration definition is created, the `trnlnkrm` command does not need to be executed after setting up OpenTP1 with the `dcsetup` command. For details on how to use the `trnlnkrm` command, see the manual *OpenTP1 Operation*. For details on how to specify the extended RM registration definition, see the manual *OpenTP1 System Definition*.

**(b) Linking**

To create executable files for a UAP, you must link the object files used for transaction control with the DBMS libraries and object modules.

Use the `trnmkobj` command to make object files used for transaction control. For details on the `trnmkobj` command, see the manual *OpenTP1 Operation*.

**(c) Writing system definitions**

To use DBMSs, you must use `trnstring` in the transaction service definition and, if necessary, use `trnrmid` in the user service definition or user service default definition. Specified contents include the items for DBMSs. For details on such items, see the appropriate manuals for the database you use.

For details on definitions using `trnstring` and `trnrmid`, see the manual *OpenTP1 System Definition*.

**(d) Setting environment variables**

Some DBMS may require special environment variables for use with OpenTP1 UAPs. If they are necessary, you must use `putenv` in the transaction service definition, user service definition, or user service default definition.

For details on definitions using `putenv`, see the manual *OpenTP1 System Definition*.

**(e) Extending the OpenTP1 internal thread stack area**

To use DBMSs, in the `thread_stack_size` operand in the transaction service definition, you must specify the size of the thread stack area to be used by OpenTP1 internally.

For details on definitions, see the manual *OpenTP1 System Definition*.



## Chapter

---

# 5. Overview of Setup, Use, and Error Recovery

---

This chapter gives an overview of the actions taken when setting up or using OpenTP1, and also describes some OpenTP1 features concerning error detection and recovery.

- 5.1 Setting up an OpenTP1 system
- 5.2 Operating an OpenTP1 system
- 5.3 Failure and error recovery

---

## 5.1 Setting up an OpenTP1 system

---

Setting up the OpenTP1 system environment requires that someone be assigned to be the *OpenTP1 administrator*. The OpenTP1 administrator is a user who can configure and manage an OpenTP1 system, and is assigned by a superuser. The set-up procedure is carried out by this superuser and the OpenTP1 administrator. If for some reason an OpenTP1 administrator is unavailable, the superuser can also act as the OpenTP1 administrator, and he or she can carry out the whole procedure by first logging into the environment as the superuser, and later as the OpenTP1 administrator. This section describes the environment setting procedures needed before starting OpenTP1. For further details, see the manual *OpenTP1 Operation* and the manual *OpenTP1 System Definition*.

In addition to the basic environment setting procedures, facility-specific procedures are needed when:

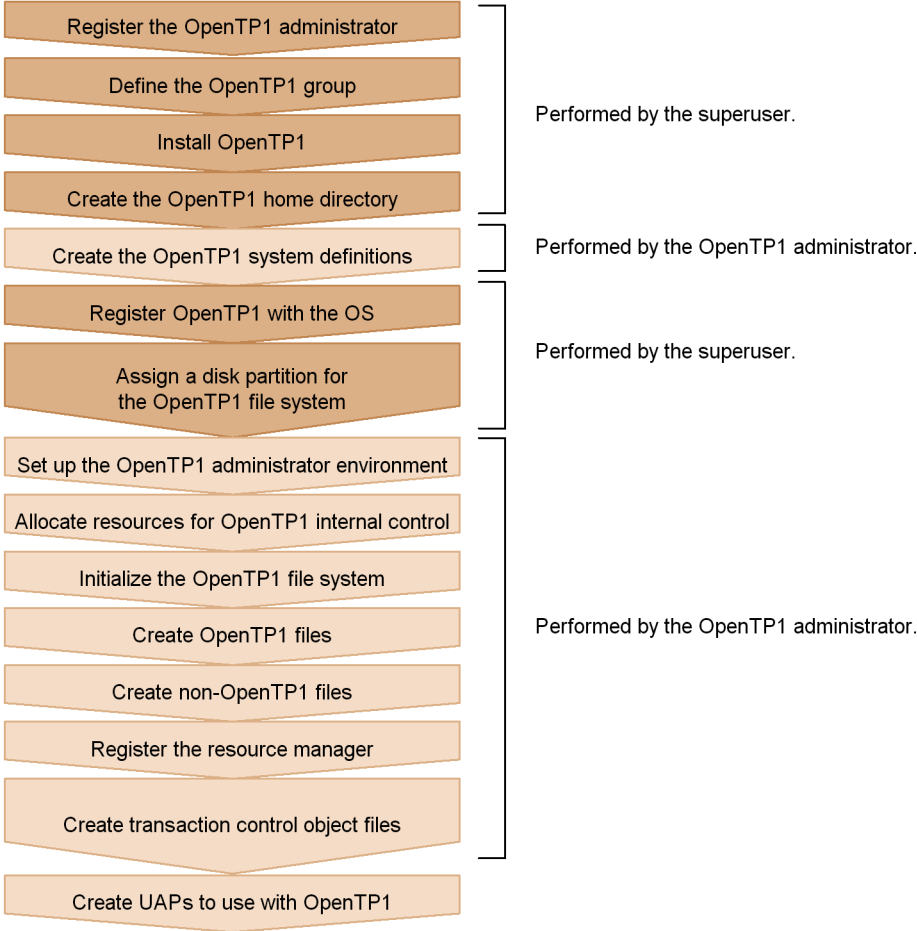
- Using the Security facility
- Using the MCF Message Exchange facility
- Using the MQA Message Queuing facility

### 5.1.1 Overview of environment settings

#### (1) *OpenTP1 environment setup*

The following figure shows the procedure for setting up OpenTP1.

Figure 5-1: OpenTP1 environment setup

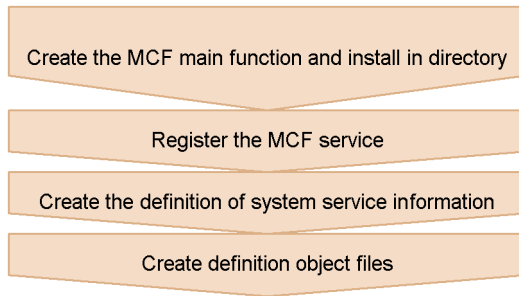


Part of OpenTP1 environment setup can be automated by using Scale Out scenario templates in collaboration with JP1/Base, JP1/AJS2, and JP1/AJS2 - Scenario Operation. For details about OpenTP1 environment setup using scenario templates, see the description of Scale Out scenario templates in the manual *OpenTP1 Operation*.

**(2) Environment setup for the message exchange facility**

The following figure shows the environment setup for the message exchange facility.

Figure 5-2: Environment setup for the message exchange facility



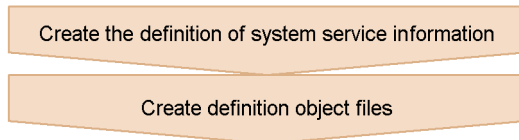
**(3) Environment setup for the message control facility**

For the environment setup for TP1/Messaging, see TP1/Messaging documentation.

**(4) Environment setup for the message queuing facility**

The following figure shows the environment setup for the message queuing facility.

Figure 5-3: Environment setup for the message queuing facility



**5.1.2 Environment setup tasks**

This section describes the tasks performed when setting up the OpenTP1 environment. For details, see the manual *OpenTP1 Operation*.

**(1) Environment setup tasks performed by the superuser**

**(a) Register the OpenTP1 administrator**

Set a user ID and associated password for the OpenTP1 administrator. Note that the OpenTP1 system does not support large user IDs. Do not use a large user ID to register the OpenTP1 administrator.

**(b) Define the OpenTP1 group**

In the group ID for OpenTP1 administrators, define which members are to belong to the OpenTP1 administrator group.

**(c) Install OpenTP1**

Install the OpenTP1 products required for the system.



**(d) Create the OpenTP1 directory**

Define directories for installing OpenTP1 products. Set the OpenTP1 administrator as the owner of the directories, and set OpenTP1 administrator group members as the owner group of the directories.

**(e) Register OpenTP1 with the OS**

Use the OpenTP1 command `dcsetup` to register OpenTP1 products with the OS.

**(f) Assign a disk partition for the OpenTP1 file system**

Assign disk or disk partitions to build an OpenTP1 file system on a character special file. If you build the OpenTP1 system file on ordinary files, this assignment is not required.

**(2) Environment setup tasks performed by the OpenTP1 administrator****(a) Create the OpenTP1 system definitions**

Create the files for the definitions used to define OpenTP1 system services, and write these system definitions. You can check the created definitions by executing the `dcdefchk` command.

**(b) Set the OpenTP1 administrator environment**

Set environment variables for the OpenTP1 administrator's login environment. Specify a directory name consisting of 50 bytes or less for the `DCDIR` environment variable.

**(c) Allocate resources for OpenTP1 internal control**

Reserve the resources to be used for OpenTP1. To reserve resources, execute the `dcmakeup` command. The amount of resources to be reserved by the `dcmakeup` command is automatically resolved according to the system service definition.

**(d) Initialize the OpenTP1 file system**

Use the OpenTP1 command `filmkfs` to initialize the area for the OpenTP1 file system. At this time, specify the OpenTP1 file system area name, using 49 or fewer characters.

**(e) Create OpenTP1 files**

Use OpenTP1 file commands, such as `jnlinit` or `stssinit`, to format the area for the OpenTP1 file system.

**(f) Create non-OpenTP1 files**

Create required ordinary files, such as executable files for UAPs or OpenTP1 definition files.

**(g) Register the resource manager**

If a resource manager other than the OpenTP1 resource manager is required, use the

command `trnlkrm` to register the resource manager in the system. If an extended RM registration definition is created, the `trnlkrm` command does not need to be executed.

**(h) Create transaction control object files**

Use the command `trnmkobj` to create transaction-control object files which control transactional UAPs.

**(i) Create UAPs to use with OpenTP1**

Create UAPs to use with OpenTP1 after you finish setting up the environment for TP1/Message Control if using the message exchange facility, and for TP1/Message Queue if using the message queuing facility.

**(3) Using the MCF Message Exchange facility**

The OpenTP1 administrator carries out the following tasks:

1. The OpenTP1 administrator writes the MCF main function and places it in a directory.
2. The OpenTP1 administrator registers MCF service names.
3. The OpenTP1 administrator creates system service information definition files.
4. The OpenTP1 administrator creates definition object files.

**(4) Using the MQA Message Queuing facility**

The OpenTP1 administrator carries out the following tasks:

1. The OpenTP1 administrator creates definition object files.
2. The OpenTP1 administrator or programmers create UAPs used in OpenTP1.

## 5.2 Operating an OpenTP1 system

OpenTP1 operations can be classified into:

- necessary routine operations
- operations that modify the OpenTP1 system
- other operations

This section describes these operations. For further details, see the manual *OpenTP1 Operation*.

Table 5-1 describes routine operations, Table 5-2 describes operations that modify the OpenTP1 system, and Table 5-3 describes other operations.

*Table 5-1: Routine operations in an OpenTP1 system*

Operation	Purpose	Procedure
Starting OpenTP1	To start operation. OpenTP1 can be started from the initial state, or restarted from its state at the previous termination.	In the system environment definition, specify whether the OpenTP1 system is to be started manually with the command <code>dcstart</code> , or started automatically whenever the OS starts.
Normal termination	To terminate operation normally.	You use the command <code>dcstop</code> to terminate OpenTP1.
Forced normal termination	Terminating OpenTP1 in a forced normal termination terminates an online system normally even if a UAP has terminated abnormally during online processing.	Use the <code>dcstop -n</code> command.
Planned termination (mode A)	Terminating OpenTP1 in a planned termination (mode A) halts the online system temporarily. For example, when an error occurs on a terminal managed by MCF, OpenTP1 terminates and leaves messages in the output queue.	Use the <code>dcstop -a</code> command.
Planned termination (mode B)	Terminating OpenTP1 in a planned termination (mode B) quickly terminates the online system. OpenTP1 terminates immediately after completion of the currently executing service.	Use the <code>dcstop -b</code> command.

Operation	Purpose	Procedure
Forced termination	Terminating OpenTP1 in a forced termination immediately terminates the online system. OpenTP1 terminates immediately without waiting for the completion of the currently executing service.	Use the <code>dcstop -f</code> command.
Starting a UAP	To start running a UAP.	Define UAPs that are to start automatically with OpenTP1 in the user service structure definition.
Terminating a UAP	To stop running a UAP. The UAP can be terminated normally or forcibly.	The UAP terminates automatically when OpenTP1 terminates. You can also use the command <code>dcsvstop</code> to terminate a UAP online.
Unloading a journal	Journals are unloaded when using the journal maintenance facility and to prepare against an error on a DAM or TAM file.	To unload a journal, use the command <code>jnlunlfg</code> to copy the system journal file to a file.
Backing up DAM and TAM files	DAM and TAM files should be backed up in case of an error in the files.	To back up DAM and TAM files, use commands to copy the DAM and TAM files.
Outputting the message log to a file	To use the messages output from OpenTP1 or UAPs.	You can output the messages stored in the message log file to the standard output.

Table 5-2: Operations that modify an OpenTP1 system

Operation	Purpose	Procedure	OpenTP1	
			Stopped	Running
Modifying a definition	To modify the configuration or execution environment of an OpenTP1 system.	Change the contents of a definition file. <ul style="list-style-type: none"> <li>If the superuser changes the definition Delete OpenTP1 from the operating system using the <code>dcsetup -d</code> command after changing the definition and terminating OpenTP1 normally. Then, execute the <code>dcsetup</code> command to re-register OpenTP1 in the operating system.</li> <li>If the OpenTP1 administrator changes the definition Execute the <code>dcreset</code> command after changing the definition and terminating OpenTP1 normally. Note that the <code>dcreset</code> command cannot be executed while operating OpenTP1.</li> </ul>	Y	N

Operation	Purpose	Procedure	OpenTP1	
			Stopped	Running
Replacing a UAP while OpenTP1 is running	To replace a UAP while OpenTP1 is running. (The new UAP will start when OpenTP1 restarts.)	Terminate the active UAP and then replace it. If necessary, the user service definition can also be replaced after the active UAP terminates normally. Then start the new UAP. If you replace the definitions without first terminating the active UAP, operation of the UAP is not guaranteed.	--	Y
Replacing a UAP while OpenTP1 is running (changing to a UAP in another directory)	To temporarily replace a UAP while OpenTP1 is running. (The UAP in the specified directory takes effect when OpenTP1 is restarted after termination.)	Terminate the active UAP and then change the search path using an operation command. After changing the search path, start the new UAP.	--	Y
Changing file capacity	To change the size of a file. (OpenTP1 must be stopped except when changing the size of a status file.)	1. Allocate a physical file with an increased capacity. 2. Redefine the file. or 1. Create a new file with a sufficient capacity. 2. Define the file. The size of a status file can be changed using an operation command.	Y	Y*
Changing the network configuration while online	You can add or delete nodes in the network of the OpenTP1 system while online.	Online reconfiguration can be achieved in either of the following ways. Method 1 (Reconfiguration using the system common definition): 1. Change the specification of the <code>all_node</code> operand in the system common definition. 2. Execute the <code>namndchg</code> command. When the <code>namndchg</code> command is executed with the <code>-l</code> option, the specification of the current <code>all_node</code> operand can be output to the standard output. Method 2 (Reconfiguration using domain definition files): 1. Create the <code>all_node</code> and <code>all_node_ex</code> domain definition files. 2. Execute the <code>namndchg</code> command.	N	Y

Operation	Purpose	Procedure	OpenTP1	
			Stopped	Running
Changing the main() function, UOCs, and libraries of the MCF communication service while OpenTP1 is running	You can reconfigure the MCF communication service without stopping OpenTP1.	To reconfigure the MCF communication service, use an operation command to partially stop the MCF communication service, and then change the main() function, UOCs, and libraries as you want.	N	Y

Legend:

Y: The task can be performed.

N: The task cannot be performed.

Y\*: The task can be performed in part.

--: Not applicable.

Table 5-3: Other operations in an OpenTP1 system

Operation	Purpose	Procedure
Monitoring status	To monitor the status of OpenTP1.	Use the relevant command to output the OpenTP1 status information to the standard output.
Using the MultiOpenTP1	MultiOpenTP1 is useful when testing OpenTP1 with a UAP. Two OpenTP1 instances are activated on the same processor: one OpenTP1 instance is for the test and the other is for practical use.	To distinguish between the two OpenTP1 instances, assign each instance a name (an OpenTP1 identifier).

---

## 5.3 Failure and error recovery

---

OpenTP1 recovery can be classified into:

- partial recovery
- complete recovery

In *partial recovery*, OpenTP1 recovers a UAP by performing a commit or rollback operation (usually a rollback) on incomplete transactions. Partial recovery localizes the effects of the UAP failure to the single UAP, and preserves the integrity of the resources used by that UAP.

In *complete recovery*, OpenTP1 recovers from a complete system failure. This requires a partial recovery for every UAP executing when the system went down, as well as recovery of the various OpenTP1 system statuses.

For further details, see the manual *OpenTP1 Operation*.

### 5.3.1 Recovering from OpenTP1 system failures

In the event of a system failure, the OpenTP1 system can be restored from historical information to its state immediately before the failure, preserving its previous online status. This start mode is known as a *restart*. OpenTP1 collects historical information in journal files in case a restart is needed.

#### (1) *Restart at complete recovery*

In the event of an OpenTP1 system failure, a *complete recovery* can restart OpenTP1 and return the whole system to the state immediately before the failure. A complete recovery uses the *history information* stored by OpenTP1. This history information records the previous online state before the failure occurred. As described in *4. File System*, an OpenTP1 system preserves its history information in journal files in preparation for a complete-recovery *restart*. During such a restart, OpenTP1 recovers components of the system in the following order:

1. System recovery
2. Service recovery
3. Transaction recovery

Online processing can be restarted after OpenTP1 completes system recovery and service recovery.

1. System recovery

During this initial stage of a complete recovery, OpenTP1 uses the information in status files. Status files contain *system control information*: such as information about the configuration of system services and UAPs, and system file

information. Using this system control information, OpenTP1 first performs a system recovery to recover those system statuses that do not depend on synchronization point processing of a transaction. OpenTP1 uses the system control information to determine which checkpoint dump or which system journal should be used for recovery.

2. Service recovery

After using the system control information to recover the status of the system, OpenTP1 starts recovering system services. System services are recovered using the data stored in checkpoint dump files and system journal files. To recover system services, OpenTP1 attempts to use the checkpoint dump of the most recent generation and all the recovery journals obtained after that checkpoint dump. If the checkpoint dump of the most recent generation is unavailable, OpenTP1 uses the next most recent generation for the recovery and all the recovery journals obtained after that checkpoint dump. Only the most recent and the next most recent generations are guaranteed.

3. Transaction recovery

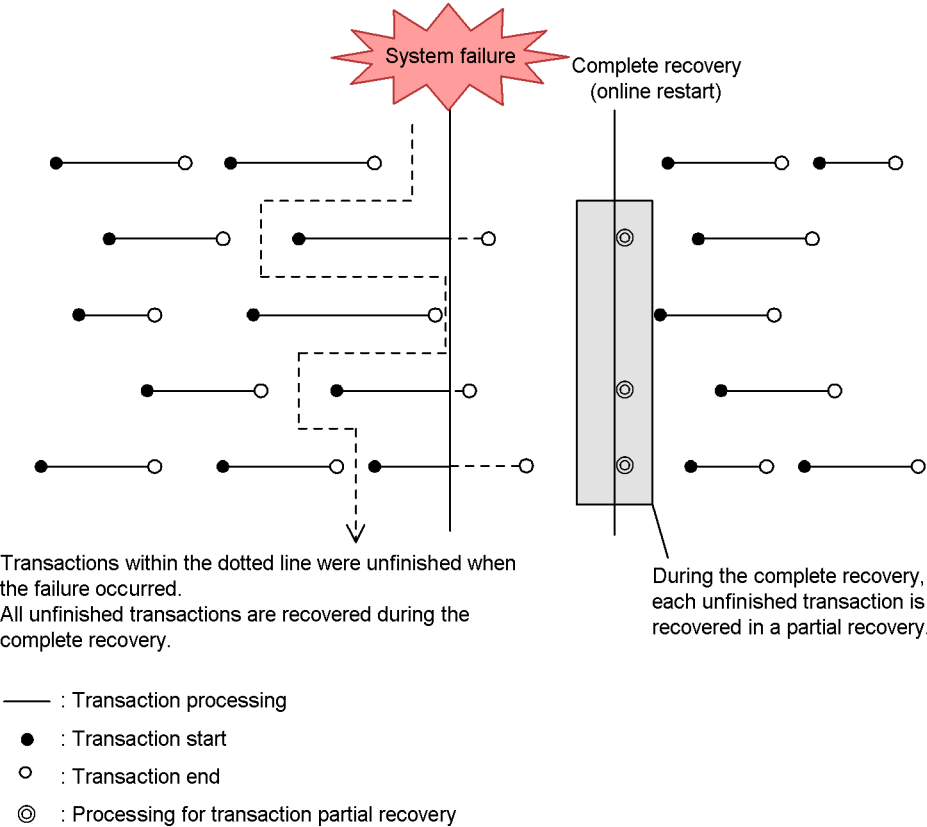
Along with the new online processing, OpenTP1 also recovers the transactions of the UAPs that were executing when OpenTP1 terminated. In this recovery, OpenTP1 performs a commit operation or rollback operation on every unfinished transaction. In effect, during this transaction recovery stage of a complete recovery, OpenTP1 recovers each UAP (that is, performs a *partial recovery* for each UAP) by recovering each unfinished transaction affected by the system shutdown.

Whether a commit or a rollback operation is performed on the transaction depends on how far the transaction processing proceeded. If the transaction is still before or at the first phase at a synchronization point, OpenTP1 rolls back the global transaction. If the transaction is already at the second phase or later, the commit or rollback operation depends on the determination of the root transaction branches.

OpenTP1 uses the synchronization point journal in the system journal files to determine how far transaction processing had proceeded when the OpenTP1 system shutdown.



Figure 5-4: Transactions recovered in a complete recovery

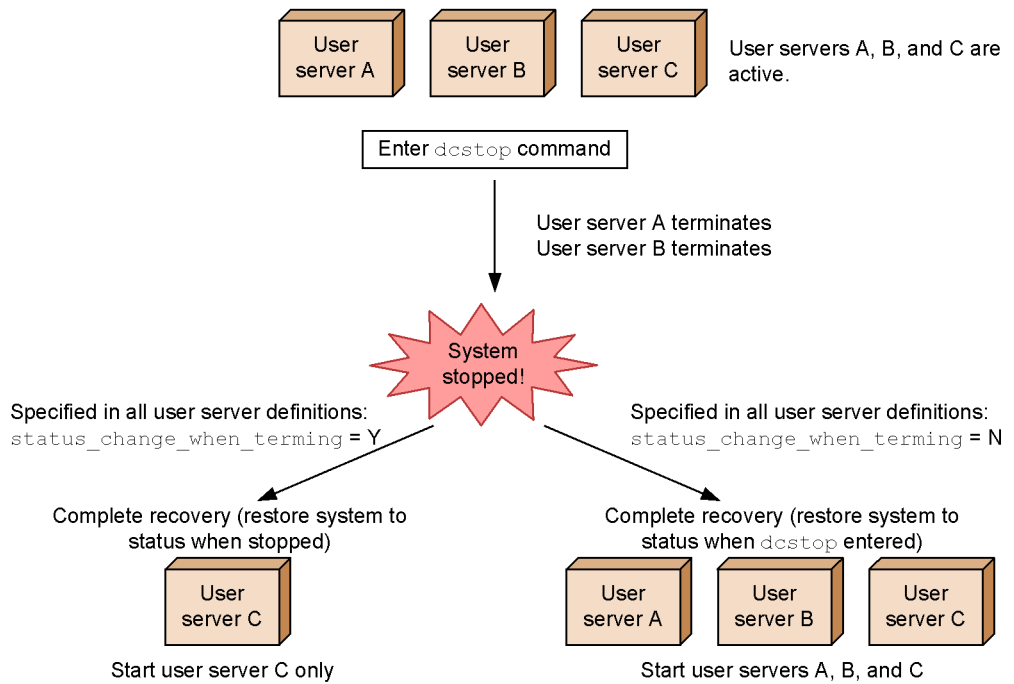


**(2) User server status at complete recovery after the OpenTP1 system stops during termination processing**

If the system stops while terminating the OpenTP1 system by entering the `dcstop` command, the user service definition and user service default definition allow you to select whether to recover the status when you input the `dcstop` command without applying the change of the user server status during the termination processing or to recover the status when the error occurred, applying the change of the status. The user server with `Y` specified by the `status_change_when_termining` operand in the user service definition applies the final status change and recovers the status when the system terminated. When specifying `N` with this operand, the user server does not apply the final status change and recovers the status when the `dcstop` command was input.

The following figure shows the status of the user servers during a complete recovery when the OpenTP1 system has stopped while in the process of terminating.

*Figure 5-5: User server status at complete recovery after the OpenTP1 system stops during termination processing*



### 5.3.2 Recovering from UAP failures

When a failure occurs in a UAP, OpenTP1 performs a partial recovery of the affected UAP, preventing the problem from impacting on the whole system. This section describes the processing when a failure occurs in a UAP.

#### (1) Recovering from an inability to start a UAP

Two common causes of being unable to start a UAP are an incorrect system definition or insufficient memory.

To recover from an incorrect system definition, the OpenTP1 administrator must correct the definitions and then restart the UAP.

To recover from insufficient memory, the OpenTP1 administrator must terminate unnecessary processes or delete unnecessary files, and then restart the UAP.

#### (2) Recovering from infinite loops in which a UAP cannot terminate

The OpenTP1 recovery procedure for an infinite loop partly depends on how OpenTP1 detected the loop. To detect a program loop, OpenTP1 monitors:

- elapsed transaction time

- RPC response time
- CPU time

OpenTP1 monitors the *elapsed transaction time*: from the start to the end of a transaction. In the user service definition, user service default definition, or transaction service definition, you can specify a limit for this time period. If this specified period is exceeded, OpenTP1 forcibly terminates the program and rolls back the transaction. This is possible for transaction processing only. If 0 is specified as the monitoring period, OpenTP1 does not monitor the elapsed time.

OpenTP1 monitors the *RPC response time*: the time elapsed from the call of the server UAP to return of control. In the system common definition or individual service definitions, you can specify a limit for this time period. If this specified period is exceeded, OpenTP1 returns an error to the source of the call. A transaction is rolled back at the synchronization point. If 0 is specified as the RPC response time, the source of the call continues to wait until it receives a response.

OpenTP1 monitors the *CPU time* in which a transaction branch can complete a transaction. In the user service definition, user service default definition, or transaction service definition, you can specify a limit for this time period. If this specified period is exceeded (i.e., the transaction branch cannot complete the whole operation within the specified period), the OpenTP1 system terminates the branch's transaction processes and executes a rollback operation.

### **(3) Abnormal termination with UAP linkage errors**

If the operating system is HP-UX, always specify `immediate` in the bind mode during linkage. If the UAP is created in a mode other than the bind mode, the UAP may terminate abnormally. Use the operating system's `chatr` command to check if the bind mode of the created UAP is `immediate`.

### **(4) Recovering from UAP abnormal terminations**

If a UAP terminates abnormally, OpenTP1 detects the abnormal termination of processing and begins *partial recovery*. In this partial recovery of a UAP transaction, OpenTP1

- restarts the UAP processing, and closes the service groups or services
- recovers the relevant transactions

#### **(a) Restarts the UAP processing, and closes the service groups or services**

What happens while restarting UAP processing and closing service groups or services depends on whether the UAP is an SPP (service-providing program), MHP (message-handling program), or SUP (service-using program).

- If an SPP terminates abnormally, OpenTP1 shuts down service groups or restarts a process depending on what is defined in the user service definition. If, however, the SPP terminates abnormally three times within the period specified for

monitoring repeated abnormal terminations, OpenTP1 shuts down the service group. An error is returned for an RPC issued to a shutdown service group.

- If an MHP terminates abnormally, OpenTP1 shuts down an MCF application or service, or restarts a process, depending on what is defined in the application attribute definition. OpenTP1 discards the message for the shutdown MHP service and issues the MCF event that reports UAP abnormal termination.
- If a SUP terminates abnormally, OpenTP1 restarts the process.

#### **(b) Recovers the relevant transactions**

If the UAP that terminated abnormally was executing a transaction, OpenTP1 performs a *transaction partial recovery*. If an error occurred in the UAP, OpenTP1 detects the abnormal termination of the UAP processing the transaction and requests the transaction recovery service to perform a partial recovery on the transaction. The transaction recovery service performs a transaction determination and recovers the transaction.

During transaction partial recovery, the transaction recovery requests are queued using the scheduling facility and the recovery is performed concurrently by multiple transaction-recovery service processes, which improves the efficiency of transaction recovery. In the transaction service definition, you can specify the number of recovery processes.

When recovery requests are queued, a transaction recovery might fail because of an OpenTP1 system area error or because of insufficient memory in the schedule queue. To successfully recover the transaction in such a case, OpenTP1 regularly checks whether an unrecovered transaction exists, and re-issues a recovery request if OpenTP1 detects an unrecovered transaction.

The preceding method is also used to perform a transaction determination that has been temporarily suppressed because a file was held by a resource manager or by a transaction service at recovery.

#### **(5) Recovering from a deadlock**

When two or more UAPs share a resource, a deadlock might occur. If it detects a deadlock, OpenTP1 compares the deadlock priorities of the UAPs and returns a lock error to the function issued from the UAP with the lowest priority.

For the OpenTP1 processing when a deadlock occurs, see *3.9.1(5) Deadlocks in TAM and DAM files*.

### **5.3.3 Recovering from file errors**

This section describes the procedure when an error occurs in one of the following OpenTP1 files:

- System file

- Queue file
- User data file

For details about recovering from file errors, see the manual *OpenTP1 Operation*.

### **(1) Recovering from system file errors**

The following describes the procedure when an error occurs in a system file.

#### **(a) Recovering from system-file errors: status files**

OpenTP1 can detect a status file error when the status information is written or read, or when the status service is started. When an error is detected while attempting to read to or write from a status file, the procedure for recovering from such an error depends on:

- whether a reserved status file exists
- whether only one or both the physical files in the current filegroup are unreadable

When an error is detected when attempting to start the status service, the system and user actions depend on which of the following is specified in the status service definition:

- OpenTP1 termination
- start of status service

#### **(b) Recovering from system-file errors: system journal files**

System journal file errors can be generally classified into write errors and read errors. In a write error, the journal to be obtained cannot be written to during online processing. In a read error, the journal cannot be read from during a complete recovery or a UAP transaction partial recovery.

The procedure for recovering from such an error depends on:

- whether the error occurred during a read or write
- whether a swappable standby filegroup exists

Note, however, that when a system journal file is duplicated, the journal is input first from physical file A. If an error occurs during the write, the journal can be input from physical file B, which increases reliability at system recovery.

When a system journal file is duplicated, even if only one of the files can be used as a swap destination, you can specify whether to use it as the swap destination in the system journal service definition.

#### **(c) Recovering from system-file errors: checkpoint dump files**

As with system journal file errors, checkpoint dump file errors can be roughly classified into write errors and read errors. In a write error, the checkpoint dump cannot

be written to during online processing. In a read error, the checkpoint file cannot be read from during a complete recovery.

If a checkpoint dump is duplicated, errors can be recovered starting from either system A or B at system recovery.

The procedure for recovering from such an error depends on:

- whether the error occurred during a read or write
- whether overwritable filegroups exist

**(d) Recovering from system-file errors: archive journal files**

Archive journal file errors can be classified into write errors and read errors. In a write error you cannot write journal information you are attempting to acquire during online processing. In a read error you cannot read journals during complete recovery.

If an archive journal file is duplicated, data is input from system A. If a write error occurs, input can be switched from system A to system B, increasing the reliability during system recovery.

When a file is duplicated, you can specify in the archive journal service definition whether to allocate the archive journal file as the swap destination even if only one standby physical file can be used.

**(e) Recovering from system-file errors: transaction recovery journal files and server recovery journal files**

If a failure occurs in a transaction recovery journal file or in a server recovery journal file, a message about the failure is output. In accordance with this message:

1. Execute the `jnlmkrf` command to restore the target recovery journal file.

If the recovery journal files are not recovered despite executing the `jnlmkrf` command:

1. Restore the resource manager using commands (`damfrc` or `tamfrc`)
2. Forcibly restart the OpenTP1 system.

**(f) Recovering from system-file errors: message log files**

For a message log file, two files are used with round-robin scheduling. If an error occurs in either of the two files, OpenTP1 uses the normal file only, isolating the erroneous file. If errors occur in both files, OpenTP1 continues processing without outputting a message log.

The procedure for recovering from such an error depends on whether OpenTP1 can detect the error:

- If a message log file error can be detected, OpenTP1 outputs error messages to the standard error output and the OpenTP1 administrator must take the required

recovery measures.

- If OpenTP1 cannot detect a message log file error, no recovery measures can be taken.

## **(2) Recovering from queue file errors**

The following describes the procedure when an error occurs in a queue file.

### **(a) Recovering from queue-file errors: MCF message queue file**

OpenTP1 cannot recover from a read or write error to an MCF message queue file. If an MCF message queue file cannot be opened, OpenTP1 outputs a message and the OpenTP1 administrator must allocate another MCF message queue file.

### **(b) Recovering from queue-file errors: MQA message queue file**

For an action taken if an error occurs on a queue file of TP1/Message Queue, see the error recovery section in the *OpenTP1 TP1/Message Queue User's Guide*.

## **(3) Recovering from user file errors**

The following describes the procedure when an error occurs in a user data file.

### **(a) Recovering from user-file errors: DAM files**

If a read or write error occurs in a DAM file, the DAM service returns an error to the UAP. When the transaction is completed, the DAM file is shutdown (i.e., access to the DAM file is prevented) because of the error.

### **(b) Recovering from user-file errors: TAM files**

When OpenTP1 detects a read or write error in a TAM file, the TAM file is shutdown: i.e., access to the TAM file is prevented.

### **(c) Recovering from user-file errors: ISAM files**

For an action taken if an error occurs on a Hitachi ISAM file, see the error recovery section in the manual *Indexed Sequential Access Method ISAM*.

## **5.3.4 Recovering from network errors**

### **(1) Error in a communication control unit, terminal, or line**

If an error occurs in a connection, OpenTP1 outputs an error message. Referring to the error message, take appropriate action to resolve the problem. Then execute the command to re-establish the connection.

The troubleshooting procedure depends on the communication protocol being used. For details on the required action, see the applicable *OpenTP1 Protocol* manual.

### **(2) LAN error**

If an error occurs on the LAN, remote procedure calls to a function on another node

will return an error to the client UAP. Restart the system after the administrator responsible for each system has taken appropriate action to recover the OS and hardware.

### (3) **Communication error in the multi-node structure**

If an error occurs between the global archive journal service and the journal service on the archive-journal source node, OpenTP1 outputs an error message. Referring to the error message, take appropriate action to resolve the problem.

No journals are archived with this type of communication error. If the error cannot be resolved and you need to unload the journals, do so separately on each archive-journal source node.

## 5.3.5 OpenTP1 monitoring and trace facilities

To detect errors in the system, OpenTP1 monitors the items listed in the table below.

*Table 5-4: Items monitored by OpenTP1*

Monitored item	Monitoring performed by OpenTP1	Relevant system definition
System initialization time	OpenTP1 monitors the time from the <code>dcstart</code> command entry until the termination of system initialization. If the system initialization does not terminate after the specified time is passed, OpenTP1 assumes a <code>user_command</code> error or another error and aborts the system startup.	System environment definition
RPC response time	OpenTP1 monitors the time elapsed from the time a service request was issued until the time a response is returned. If no response is returned within a specified period, an error is returned to the source of the request. The error is a sending/receiving timeout caused, for example, by a server UAP loop.	System common definition
Chained RPCs	OpenTP1 monitors the time elapsed from the time the server UAP returns control to a service request until the time the next RPC or request to perform a transaction determination is received. If neither the next RPC nor a transaction determination instruction is received within a specified period, OpenTP1 assumes that an error occurred at the source of the service request and abnormally terminates the server UAP process.	User service definition User service default definition.



Monitored item	Monitoring performed by OpenTP1	Relevant system definition
Deadlocks	OpenTP1 monitors the time elapsed from the time a lock request is forced to wait until the waiting ends. If the waiting period does not end within a specified period, OpenTP1 assumes that a deadlock occurred and returns an error to the source of the request.	Lock service definition
Transaction branches	OpenTP1 monitors the processing time of a transaction branch. If the processing is not completed within a specified period, OpenTP1 assumes that an error occurred in the UAP, abnormally terminates the transaction branch process, and rolls back the transaction.	User service definition User service default definition Transaction service definition.
CPU time	OpenTP1 monitors the CPU time in which a transaction branch can complete a transaction. If the transaction branch cannot complete the whole operation within a specified period, OpenTP1 abnormally terminates the branch's transaction processes and executes a rollback operation.	User service definition User service default definition Transaction service definition.
UAP repeated abnormal terminations	OpenTP1 monitors the time during which a UAP repeatedly abnormally terminates. If a UAP abnormally terminates three times within a specified period, OpenTP1 shuts down scheduling for the service group or service regardless of the shutdown specification in the user service definition.	User service definition User service default definition
Remaining service requests	OpenTP1 monitors the number of service requests staying in the schedule queue. If the number exceeds (the value specified <code>balance_count</code> operand $\times$ number of actual processes), OpenTP1 activates a non-resident process to execute the service.	User service definition User service default definition
I/O queue usage rate	physical files for each queue group of the I/O queue. If the rate exceeds a specified percentage, OpenTP1 outputs a warning message.	Message queue service definition
Number of times communication function is issued	OpenTP1 checks the number of times communication functions are issued in a transaction. This monitoring can detect endless loops in a UAP. If the number of issued functions exceeds a specified number, OpenTP1 abnormally terminates the UAP and starts a rollback.	MCF manager definition

Monitored item	Monitoring performed by OpenTP1	Relevant system definition
Lifetimes of unprocessed send messages	OpenTP1 monitors how long unprocessed messages that are to be sent can stay in the output queue. This monitoring prevents normal termination processing from taking a long time. If unprocessed messages that are to be sent are left after a specified period elapses, OpenTP1 discards them (after reporting the MCF event ERREVT) and continues normal termination processing.	MCF configuration definition

### 5.3.6 Analyzing the cause of an error

This subsection describes the functions for analyzing the causes of errors in the OpenTP1 system.

Note that the functions listed below assume that TP1/Extension 1 has been installed. If TP1/Extension 1 has not been installed, the operation of these functions cannot be guaranteed.

- Performance verification trace
- XAR performance verification trace
- JNL performance verification trace
- LCK performance verification trace
- MCF performance verification trace
- TRN event trace
- NAM event trace
- Process service event trace
- FIL event trace

#### (1) MCF trace

With the *MCF trace facility*, MCF obtains information, such as events and data sent or received, for each process. The *MCF trace information* is output to the trace area (trace buffer) in the shared memory. If the MCF trace disk output facility is specified in the MCF communication configuration definition and the trace area becomes full, the MCF trace information is output to an MCF trace file.

You might temporarily require the MCF trace information while the disk output facility is disabled. In this case, you can output the MCF trace information to an MCF trace file by using the MCF-trace-information acquisition start command (`mcftrace`) and the MCF-trace-information acquisition stop command

(`mcfststptr`). For details about the commands, see the manual *OpenTP1 Operation*.

The MCF trace facility collects information about the functioning, before an error occurred, of control functions within a process and the control flow for each event.

## (2) UAP trace

When an OpenTP1 UAP (SUP, SPP, or MHP) terminates abnormally, trace information is output from the API called by the UAP. This is called a *UAP trace*, and the acquired information is referred to as *UAP trace information*. OpenTP1 acquires UAP trace information separately for each UAP process.

UAP trace information is stored as files named *server-nameN.uat* (where *N* represents a serial number appended to the core file of the UAP process) under the `$DCDIR/spool/save/` directory. These files are known as *UAP trace edit/output files*.

UAP trace information can be output from a core file generated at abnormal termination of the UAP. To obtain the trace information, execute the `uatdump` command, specifying the required core file. The UAP trace information can be output to a specified file by redirecting the `uatdump` command's output destination.

The value specified in the `uap_trace_max` operand of the user service definition determines the maximum size of UAP trace information that can be stored in a file. When this value is reached, subsequent trace information is wrapped around to the start of the file.

UAP trace information is acquired separately for each abnormally terminated UAP process. When transaction processing called by an RPC on another node terminates abnormally, UAP trace information is acquired for each of the UAP processes executed on the two nodes.

For details about UAP traces, see description in the *OpenTP1 Tester and UAP Trace User's Guide*.

## (3) RPC trace

OpenTP1 has an *RPC trace facility* to obtain information about RPC service requests; this information is collected into a file. You can use an RPC management command `rpcdump` to dump the RPC trace information. Some of the uses of the RPC trace facility are:

- to find out which client UAP performed inter-process communication with which server UAP
- to find out the order in which a specified UAP executed services
- to analyze the flow of service requests up to the time an RPC error occurred

You can use the `rpcdump` command to edit the contents of an RPC trace file.

You can obtain an RPC trace file for each system service. To obtain the file, in the

system service definition for each service specify that the trace is to be obtained. You can use the `rpcmrgr` command to merge and print, in chronological order, the RPC trace files from several system services.

#### (4) Performance verification trace

OpenTP1 has a *performance verification trace* to acquire trace information, such as an OpenTP1 identifier, for major events of services running in OpenTP1.

The advantages of conducting a performance verification trace are:

- tracing events across nodes and processes
- acquiring a trace for an internal event, rather than for an API, to verify which process causes a bottleneck in the performance

The following table lists and describes the system definitions that are related to performance verification trace.

Table 5-5: System definitions related to performance verification trace

Definition name	Format	Operand	Description
System common definition	set	<code>prf_trace</code>	Specifies whether performance verification trace information is to be acquired.
	set	<code>trn_prf_trace_level</code>	Sets the trace acquisition level
Performance verification trace definition	set	<code>prf_file_size</code>	Sets the size of trace file

For details about the definitions, see the manual *OpenTP1 System Definition*.

You can use the `prfget` command to retrieve a performance verification trace in binary format and then use the `prfed` command to output it in character format.

You can also use the `dc_prf_utrace_put` function to collect user-specific trace information in a trace file and the `dc_prf_get_trace_num` function to obtain the sequential number of the most recent trace acquired in the process.

Notes:

- When OpenTP1 is restarted in the normal way or by a hot standby method, no trace information is carried over.
- Trace acquisition performed by the performance verification trace facility is not subject to control by locks so that performance of online processing will not be affected. For this reason, if contention occurs during trace acquisition in a multiprocessor environment, some trace information may be missing or invalid trace information may be acquired. When trace information is edited

using the `prfed` command, invalid information is displayed as error records.

### (5) XAR performance verification trace

OpenTP1 collects trace information about all events involving transactional linkage using the XA resource service, such as transaction requests from an application server and OpenTP1 transaction processing. This information is known as an *XAR performance verification trace*.

XAR performance verification traces are stored as files with the file name `_xr_001`, `_xr_002`, `_xr_003`, and so on, under the `$DCDIR/spool/dcxarinf/` directory. These files are known as *XAR trace information files for verification*. The output directory and file names cannot be changed for the XAR performance verification trace information files. You can change the size and number of trace information files that can be acquired.

For details, see the description of the XAR performance verification trace definition in the manual *OpenTP1 System Definition*.

To collect XAR performance verification trace information:

1. Set `Y` in the `prf_trace` operand in the system common definition.
2. Specify the level of XAR performance trace information to acquire in the `xar_prf_trace_level` operand in the XA resource service definition.

The following table describes the XAR performance verification trace information that is collected depending on the value of the `xar_prf_trace_level` operand.

*Table 5-6:* Relationship between the value specified in the `xar_prf_trace_level` operand and acquired XAR performance verification trace information

Value of <code>xar_prf_trace_level</code> operand	Type of XAR trace information collected	Event ID	Trace data size (bytes)	Collection timing	
00000001	Transaction request from an application server	0x4a00	128	Request to start a transaction branch	Immediately after call
		0x4a01	128		Immediately before return
		0x4a02	128	RPC execution request from within a transaction branch	Immediately after call
		0x4a03	128		Immediately before return

5. Overview of Setup, Use, and Error Recovery

Value of xar_prf_trace_level operand	Type of XAR trace information collected	Event ID	Trace data size (bytes)	Collection timing	
		0x4a04	128	Request to terminate a transaction branch	Immediately after call
		0x4a05	128		Immediately before return
		0x4a06	128	Request to prepare a transaction branch for commit	Immediately after call
		0x4a07	128		Immediately before return
		0x4a08	128	Request to commit a transaction branch	Immediately after call
		0x4a09	128		Immediately before return
		0x4a0a	128	Request to roll back a transaction branch	Immediately after call
		0x4a0b	128		Immediately before return
		0x4a0c	64	Request to report a transaction branch in Prepared or Heuristically Completed status	Immediately after call
		0x4a0d	64		Immediately before return
		0x4a0e	128	Request to discard a Heuristically Completed transaction branch	Immediately after call
		0x4a0f	128		Immediately before return
		00000002	OpenTP1 transaction process	0x4b00	64
0x4b01	64			Immediately after	
0x4b02	64			Execute an RPC from within a transaction branch	Immediately before
0x4b03	64				Immediately after

Value of <code>xar_prf_trace_level</code> operand	Type of XAR trace information collected	Event ID	Trace data size (bytes)	Collection timing	
		0x4b04	64	Terminate a transaction branch	Immediately before
		0x4b05	64		Immediately after
		0x4b06	64	Prepare a transaction branch for commit	Immediately before
		0x4b07	64		Immediately after
		0x4b08	64	Commit a transaction branch	Immediately before
		0x4b09	64		Immediately after
		0x4b0a	64	Roll back a transaction branch	Immediately before
		0x4b0b	64		Immediately after
		0x4b0c	64	Report a transaction branch in Prepared or Heuristically Completed status	Immediately before
		0x4b0d	64		Immediately after
		0x4b0e	64	Discard a Heuristically Completed transaction branch	Immediately before
		0x4b0f	64		Immediately after

For details about the `xar_prf_trace_level` operand, see the description of the XA resource service definition in the manual *OpenTPI System Definition*.

To collect XAR trace information files and output edited trace information, use the `prfget` and `prfed` commands. The acquisition and edit/output procedures are as follows.

#### Collecting XAR trace information for verification

To collect only the trace information that has not been acquired with the latest run ID, execute the following commands:

```
$DCDIR/bin/prfget -f _xr | $DCDIR/bin/prfed -d
```

To collect all trace information, execute the following commands:

```
$DCDIR/bin/prfget -a -f _xr | $DCDIR/bin/prfed -d
```

Editing and outputting XAR trace information for verification

To edit and output trace information from XAR trace information files for verification, execute the `prfed` command. Specify the options as required.

XAR trace information for verification is output in the same format as the performance verification trace. For details about the output format, see the description of the `prfed` command in the manual *OpenTPI Operation*.

### (6) JNL performance verification trace

OpenTPI collects trace information about various events related to journal buffering and journal output performed by the journal service. This information is called *JNL performance verification trace*.

JNL performance verification traces are stored as files with the file name `_j1_001`, `_j1_002`, `_j1_003`, etc., under the `$DCDIR/spool/dcjnlinf/prfinf` directory. These files are called *JNL performance verification trace information files*. The output destination and names of the JNL performance verification trace information files cannot be changed.

To collect JNL performance verification trace information:

1. Specify `Y` in the `prf_trace` operand in the system common definition.
2. Also specify in the system common definition the level of JNL performance verification trace information to be acquired. Make this specification in the `jnl_prf_event_trace_level` operand.

The table below shows the trace information that is collected depending on the specification of the `jnl_prf_event_trace_level` operand. For details about the timing of acquiring each event, see the description of the acquisition of performance verification trace information in the manual *OpenTPI Operation*.

*Table 5-7: Relationship between the `jnl_prf_event_trace_level` operand value and the trace information that is collected*

jnl_prf_event_trace_level operand value	Event IDs of trace information	
	0xc202, 0xc203, 0xc401, 0xc402	0xc001 to 0xc201, 0xc204 to 0xc400
00000000	N	N
00000001	Y	N
00000002	Y	Y
Other	Y	Y



Legend:

Y: Trace information is acquired.

N: Trace information is not acquired.

You use the `prfget` command to collect JNL performance verification trace information files and the `prfed` command to edit and output the files. The following describes how to collect the files.

How to collect JNL performance verification trace information files:

To collect only the trace information that has not been acquired with the most recent run ID, execute the following commands:

```
$DCDIR/bin/prfget -f _j1 | $DCDIR/bin/prfed -d
```

To collect all trace information, execute the following commands:

```
$DCDIR/bin/prfget -a -f _j1 | $DCDIR/bin/prfed -d
```

### **(7) LCK performance verification trace**

OpenTP1 collects trace information about locking involved in transaction processing. This information is called *LCK performance verification trace*.

LCK performance verification traces are stored as files with the file name `_lk_001`, `_lk_002`, `_lk_003`, etc., under the `$DCDIR/spool/dclckinf/prf` directory. These files are called *LCK performance verification trace information files*. The output destination and names of the LCK performance verification trace information files cannot be changed, but you can change the size and number of LCK performance verification trace information files that can be acquired. For details, see the description of the LCK performance verification trace definition in the manual *OpenTP1 System Definition*.

To collect LCK performance verification trace information:

1. Specify Y in the `prf_trace` operand in the system common definition.
2. In the `lck_prf_trace_level` operand in the lock service definition, specify the level of LCK performance verification trace information to acquire.

The table below shows the LCK performance verification trace information that is collected depending on the specification of the `lck_prf_trace_level` operand.

*Table 5-8: Relationship between the lck\_prf\_trace\_level operand value and the LCK performance verification trace information that is collected*

<b>lck_prf_trace_level operand value</b>	<b>LCK performance verification trace information</b>	<b>Event ID</b>	<b>Trace data size (bytes)</b>	<b>Collection timing</b>	
00000000	Trace information about locking is not acquired.	--	--	--	
00000001	Trace information about locking is acquired.	0x6400	128	Locking of resources	Immediately after call
		0x6401	128		Immediately before return
		0x6410	128	Lock release wait	Immediately before start
		0x6411	128		Immediately after termination
		0x6420	128	Unlocking of all resources	Immediately after call
		0x6421	128		Immediately before return
		0x6430	128	Unlocking with a resource name specified	Immediately after call
		0x6431	128		Immediately before return

For details about the `lck_prf_trace_level` operand, see the description of the lock service definition in the manual *OpenTPI System Definition*.

You use the `prfget` command to collect LCK performance verification trace information files and the `prfed` command to edit and output the files. The following describes how to collect and how to edit and output the files.

How to collect LCK performance verification trace information files:

To collect only the trace information that has not been acquired with the most recent run ID, execute the following commands:

```
$DCDIR/bin/prfget -f _lk | $DCDIR/bin/prfed -d
```

To collect all trace information, execute the following commands:

```
$DCDIR/bin/prfget -a -f _lk | $DCDIR/bin/prfed -d
```

How to edit and output LCK performance verification trace information files:

To edit and output trace information in LCK performance verification trace information files, execute the `prfed` command with options specified, as necessary.

The output format of LCK performance verification trace information is the same as for the performance verification trace. For details about the output format, see the `prfed` command in the manual *OpenTPI Operation*.

### (8) MCF performance verification trace

OpenTPI collects trace information (such as the MCF identifier) about the main events involving message transmission using TP1/Message Control. This information is called *MCF performance verification trace*.

The MCF performance verification trace provides the following advantages:

- You can output MCF-specific details, such as thread ID, logical terminal name, and API name.
- If you analyze the MCF performance verification trace by using the detailed information that was output as the key, you can verify the performance of a series of message transmission processing transactions and UAP.

The following table lists and describes the system definitions that are related to the MCF performance verification trace.

*Table 5-9: System definitions related to the MCF performance verification trace*

Definition name	Format	Operand	Description
User service definition	set	<code>mcf_prf_trace</code>	Specifies whether MCF performance verification trace information is acquired for each user server
MCF performance verification trace definition	set	<code>prf_file_size</code>	Size of a trace file for the MCF performance verification trace information
	set	<code>prf_file_count</code>	Number of trace file generations for the MCF performance verification trace information
Definition of system service information	set	<code>mcf_prf_trace</code>	Specifies whether MCF performance verification trace information is acquired for each MCF communication service
System service common information definition	set	<code>mcf_prf_trace_level</code>	Acquisition level of MCF performance verification trace information

For details about the definitions, see the manual *OpenTP1 System Definition*.

You can use the `prfget` command to retrieve an MCF performance verification trace in binary format and then use the `prfed` command to output it in character format.

#### Notes

- The default is that the MCF performance verification trace is not collected. To collect it, you must specify `Y` in the `prf_trace` operand or omit this operand in the system common definition and also specify `00000001` in the `mcf_prf_trace_level` operand in the system service common information definition.
- You can collect the MCF performance verification trace during message transmission (event IDs `0xa000` and `0xa001`) only when you use one of the following protocol products:
  - TP1/NET/TCP/IP
  - TP1/NET/XMAP3
  - TP1/NET/OSAS-NIF

For events other than message transmission (event IDs other than `0xa000` and `0xa001`), you can collect the MCF performance verification trace regardless of the type of communication protocol.

### (9) XAR event trace

The XAR event trace acquires the type of the transaction request sent from an application server using the XA resource service as event trace information. The acquired information is called the *XAR event trace information*.

The following table lists the types of transaction requests sent from an application server, request codes, and their meaning.

*Table 5-10: Types of transaction requests and request codes*

Request type <sup>#</sup>	Request code	Meaning of request code
<code>Start()</code>	<code>xar_start</code>	Starts a transaction branch.
<code>Call()</code>	<code>xar_call</code>	Executes an RPC from within a transaction branch.
<code>End()</code>	<code>xar_end</code>	Ends a transaction branch.
<code>Prepare()</code>	<code>xar_prepare</code>	Prepares the commit for a transaction branch (first phase of a two-phase commit).
<code>Commit()</code>	<code>xar_commit</code>	Commits a transaction branch (second phase of a two-phase commit).
<code>Rollback()</code>	<code>xar_rollback</code>	Rolls back a transaction branch.

Request type <sup>#</sup>	Request code	Meaning of request code
Recover()	xar_recover	Notifies a transaction branch in the Prepared status or the Heuristically Completed status.
Forget()	xar_forget	Discards a transaction branch in the Heuristically Completed status.

#

Transaction request types are internal functions of OpenTP1.

The XAR event trace information is acquired in the `xarevtr1` or `xarevtr2` file under the `$DCDIR/spool/dcxarinf/trace/` directory. These files are called the *XAR event trace information files*. You cannot change the output destinations and files names of XAR event trace information files.

If there is an existing XAR event trace information file when you start the XA resource service, a backup file is created for the existing file and another, unused XAR event trace information file is prepared as the output destination. For example, if the `xarevtr1` XAR event trace information file exists, the file is renamed to `xarevtr1.bk1`. Because this renaming causes the file name `xarevtr1` to become available, a new XAR event trace information file is then created with the name of `xarevtr1`. Up to three generations of backup files are created. Backup files are created only when there is already an XAR event trace information file at the start of the XA resource service. During online processing, if the number of records written to one file exceeds the value of the maximum number of records that can be output to an XAR event trace information file, then no backup file is created. This maximum number is specified in the `xar_eventtrace_record` operand in the XA resource service definition. If one output file becomes full, the next file is used. If all the files have become full, the oldest one is overwritten with new records. Do not delete the XAR event trace information files while the system is online.

You can specify the output level of the XAR event trace information by using the `xar_eventtrace_level` operand in the XA resource service definition. By default, the output level is specified to acquire the XAR event trace only if an error occurs. You can acquire all the XAR event trace information if you want to. However, in that case, the online performance is affected. We recommend that you use the default output level except for debugging.

You can use the `xarevtr` command to edit and display the XAR event trace information acquired in the output file.

For details on how to specify the output level of the XAR event trace information and the number of output records, see the manual *OpenTP1 System Definition*. For details on how to edit and display the XAR event trace information, see the manual *OpenTP1 Operation*.

**(10) TRN event trace**

The *TRN event trace* is historical information about the XA functions issued in transaction branches, and about the events of transaction services (transaction management service, transaction recovery service, and resource manager monitoring service).

The TRN event trace is stored as files named `_tr_nnn` (*nnn*: 3-digit number starting at 001) in the `$DCDIR/spool/dctrninf/trace/prf` directory. These files are called *TRN event trace information files*. You cannot rename or move these files.

You can change the size of each TRN event trace information file and the maximum number of TRN event trace information files that can be stored. For details, see the TRN event trace definition in the manual *OpenTPI System Definition*.

To collect the TRN event trace information:

1. Set `Y` in the `prf_trace` operand in the system common definition.
2. Use the `trn_prf_event_trace_level` operand in the transaction service definition to specify the level for collecting the TRN event trace information.
3. Use the `trn_prf_event_trace_condition` operand in the transaction service definition to specify the type of TRN event trace collection.

The following table lists the values that can be specified in the `trn_prf_event_trace_condition` operand and the types of TRN event trace collection specified by the values.

The table below shows the TRN event trace information that is collected depending on specification of the `trn_prf_event_trace_condition` operand.

*Table 5-11:* Relationship between the `trn_prf_event_trace_condition` operand value and the TRN event trace information that is collected

Value of <code>trn_prf_event_trace_condition</code>	TRN event trace information that is collected	Event ID	Trace data size (bytes)	Collection timing
<code>xafunc</code>	Trace information about XA functions	0x4500	320 (192 for the <code>xa_open</code> and <code>xa_close</code> functions)	During transaction processing <sup>#</sup>
<code>trnservice</code>	Trace information about the operating status of the transaction service	0x4501	192	When the transaction service starts, ends, or is recovered

#

For a transaction committed in two phases, the trace amount collected per transaction branch is  $12 \times$  number of resource managers. However, the trace amount varies depending on the conditions, such as the XA interface object files linked to the user server and the transaction optimization settings.

For details about this operand, see the manual *OpenTPI System Definition*.

To collect TRN event trace information files and output edited trace information, use the `prfget` and `prfed` commands. The acquisition and edit/output procedures are as follows.

#### Collecting TRN event trace information

To collect only the trace information that has not been acquired with the latest run ID, execute the following commands:

```
$DCDIR/bin/prfget -f _tr | $DCDIR/bin/prfed -d
```

To collect all the trace information, execute the following commands:

```
$DCDIR/bin/prfget -a -f _tr | $DCDIR/bin/prfed -d
```

#### Outputting TRN event trace information

To output TRN event trace information, execute the `prfed` command with the `-d` option specified. Specify other options as required.

TRN event trace information is output in the same format as the performance verification trace. For details about the output format, see the description of the `prfed` command in the manual *OpenTPI Operation*.

The following shows an example of TRN event trace information output when `xafunc` is specified in the `trn_prf_event_trace_condition` operand.

```
PRF: Rec Node: trn1 Run-ID: 0x4046b806 Process: 26264
Trace: 10
Event: 0x4500 Time: 2004/01/01 12:34:56 678.123.000
Server-name: Sup
Rc: 0 Client: **** - ***** Server: **** Root:
trn1 - *****
Svc-Grp: ***** Svc:
*****
Trn: 78d0trn100000001trn1trn100000001
      xa_commit (IN) OpenTP1_TAM
      axid:0100000000000000c00000010

3738643074726e330000000174726e3374726e330000000100000000
Internal code1:0x2007 Internal code2: 0
Internal code3:0
```

**(11) NAM event trace**

OpenTP1 collects trace information about the various events generated by the name service, including communication invoked by the name service and the registration and deletion of service information from the cache. This information is known as an *NAM event trace*.

NAM event traces are stored as files with the file name `_nm_001`, `_nm_002`, `_nm_003`, and so on, under the `$DCDIR/spool/dnaminf/` directory. These files are known as *NAM event trace information files*. The output directory and file names cannot be changed.

To collect NAM event trace information:

1. Set `Y` in the `prf_trace` operand in the system common definition.
2. Specify the level of NAM event trace information to acquire in the `nam_prf_trace_level` operand in the system common definition.

The following table lists the values that can be specified in the `nam_prf_trace_level` operand and the types of NAM event trace information acquired for each level. For the collection timing for each event, see the description of collecting performance verification trace information in the manual *OpenTP1 Operation*.

*Table 5-12:* Relationship between the value specified in the `nam_prf_trace_level` operand and acquired NAM event trace information

Value of <code>nam_prf_trace_level</code> operand	Event ID of trace information		
	0xf000 to 0xf029	0xf100 to 0xf109	0xf200 to 0xf218
00000000	N	N	N
00000001	N	Y	N
00000002	Y	N	N
00000003	Y	Y	N
00000004	N	N	Y
00000005	N	Y	Y
00000006	Y	N	Y
00000007	Y	Y	Y
Other value	Y	Y	N

Legend:



Y: Trace information is acquired.

N: Trace information is not acquired.

To collect NAM event trace information files and output edited trace information, use the `prfget` and `prfed` commands. The acquisition and edit/output procedures are as follows.

To collect NAM event trace information, execute the following commands:

```
$DCDIR/bin/prfget -a -f _nm | $DCDIR/bin/prfed -d
```

### (12) *Process service event trace*

OpenTP1 collects information about process events such as process creation, destruction, activation, and termination. This information is known as a *process service event trace*.

Process service event traces are stored as files with the file name `_pr_001`, `_pr_002`, `_pr_003`, and so on, under the `$DCDIR/spool/dcprcinf/` directory. These files are known as *process service event trace information files*. The output directory and file names cannot be changed.

To collect process service event traces, set Y in the `prc_prf_trace` operand in the process service definition.

For the collection timing for each event, see the description of collecting performance verification trace information in the manual *OpenTP1 Operation*.

To collect process service event trace information files and output edited trace information, use the `prfget` and `prfed` commands. The acquisition and edit/output procedures are as follows.

To collect process service event trace information, execute the following commands:

```
$DCDIR/bin/prfget -a -f _pr | $DCDIR/bin/prfed -d
```

### (13) *FIL event trace*

OpenTP1 collects event information when the amount of time it takes to process an OpenTP1 file access request issued internally from a process under OpenTP1 control exceeds the value specified in the `fil_prf_trace_delay_time` operand in the system common definition. This is called a *FIL event trace*.

By analyzing the FIL event trace, you can check the delay status of processing related to OpenTP1 file access.

FIL event traces are stored as files with the file name `_fl_001`, `_fl_002`, `_fl_003`, etc., under the `$DCDIR/spool/dcfilinf/` directory. These files are called *FIL event trace information files*. The output destination and names of the FIL event trace information files cannot be changed.

You use the `prfget` command to collect FIL event trace information files and the

`prfed` command to edit and output the files. The following describes how to collect the files.

How to collect FIL event trace information files:

```
$DCDIR/bin/prfget -a -f _f1 | $DCDIR/bin/prfed -d
```

#### **(14) Command logs**

Every time an OpenTP1 operation command is executed, information such as the command start time and command termination time is output to `cmdlog1` and `cmdlog2` in `$DCDIR/spool/cmdlog`.

You can open `cmdlog1` and `cmdlog2` with a text editor such as `vi`. From the logged command start time and command termination time, you can obtain, for example, the time required to execute the command (response time).

## Chapter

---

# 6. Using Multiple Instances of OpenTP1

---

This chapter describes the facilities for implementing OpenTP1 in a large-scale system (the System Switchover facility and multi-node facility) and the support available when using a multiOpenTP1 configuration or multi-homed host configuration.

- 6.1 The System Switchover facility
- 6.2 The Multinode facility
- 6.3 The MultiOpenTP1
- 6.4 Multi-homed host configuration

---

## 6.1 The System Switchover facility

---

You can duplicate an OpenTP1 system to improve reliability. This type of duplicated system is sometimes called a *hot-standby system* or a *system-switch system*.

Duplicating an OpenTP1 system requires the product HAmonitor. For further details on HAmonitor, see the relevant HAmonitor manuals.

### 6.1.1 Overview of the System Switchover facility

OpenTP1 enables an OpenTP1 administrator to construct an OpenTP1 system consisting of two duplicated server systems. Each server system consists of OpenTP1, a CPU, and a kernel. When one of the system fails, OpenTP1 can automatically switch to the duplicated system. The parts of OpenTP1 that enable this switching are collectively called the *System Switchover facility*.

After the system switches, you can automatically or manually restart the OpenTP1 system in which an error occurred and prepare for another system switch.

The system which handles work processing online is called the *running system*, and the other system is called the *standby system*. Whenever a System Switchover occurs during online processing, the standby system becomes the new running system, and the old running system becomes the new standby system.

The system can be switched even if the Message Control facility (MCF) or the Message Queuing facility (TP1/Message Queue) is installed in the OpenTP1 system.

#### (1) Types of system switching

There are two types of system switching:

- automatic system switchover
- planned system switchover

In an automatic system switchover, the running system that has an error is automatically switched to the standby system. In a planned system switch, the running system that has an error is switched using an HAmonitor command.

In both types of switchover, you can treat several products as a group for switching. This is called *grouped system switchover*. In this type of switchover, when an error occurs in one of the products in a group of a running system, all the products in the group are switched to the standby system. In this operation, you can always use a group of products in the same running system together.

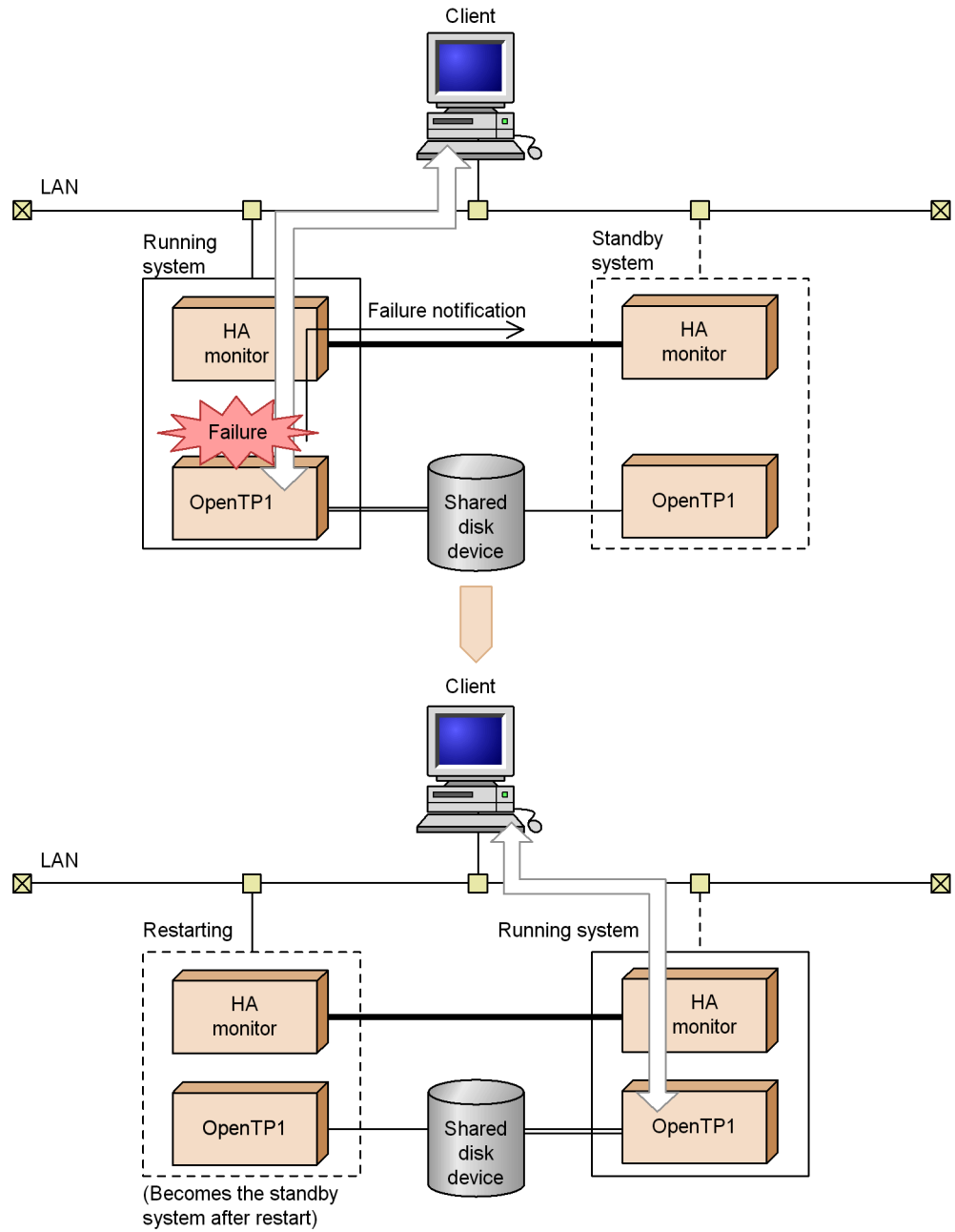
In a grouped system switchover, the timing of system switching may differ from the timing when the products are not grouped depending on the definition of HAmonitor. For details of the grouped system switchover, see the relevant HAmonitor manuals.

**(2) Identifying the primary and secondary systems**

During the setup of HAmonitor, you need to decide which system starts as the running system in an online start. You must define the system which starts first as the *primary system*, and define the standby system ready for an error of the running system as the *secondary system*. The systems keep these names even after a system switch.

The following figure provides an overview of the System Switchover facility.

Figure 6-1: Overview of the System Switchover facility



## 6.1.2 OpenTP1 system configuration for using the System Switchover facility

The following software products are required on both the primary and secondary OpenTP1 systems (TP1/Server Base) when using the System Switchover facility:

- HAmonitor, TP1/High Availability

These are required whenever the System Switchover facility is used.

- TP1/NET/High Availability

This is required for using the System Switchover facility and the message control facilities (TP1/Message Control and TP1/NET/Library).

The following hardware is used in common by the primary and secondary systems:

- Shared disk device

Shared disk device

- LAN

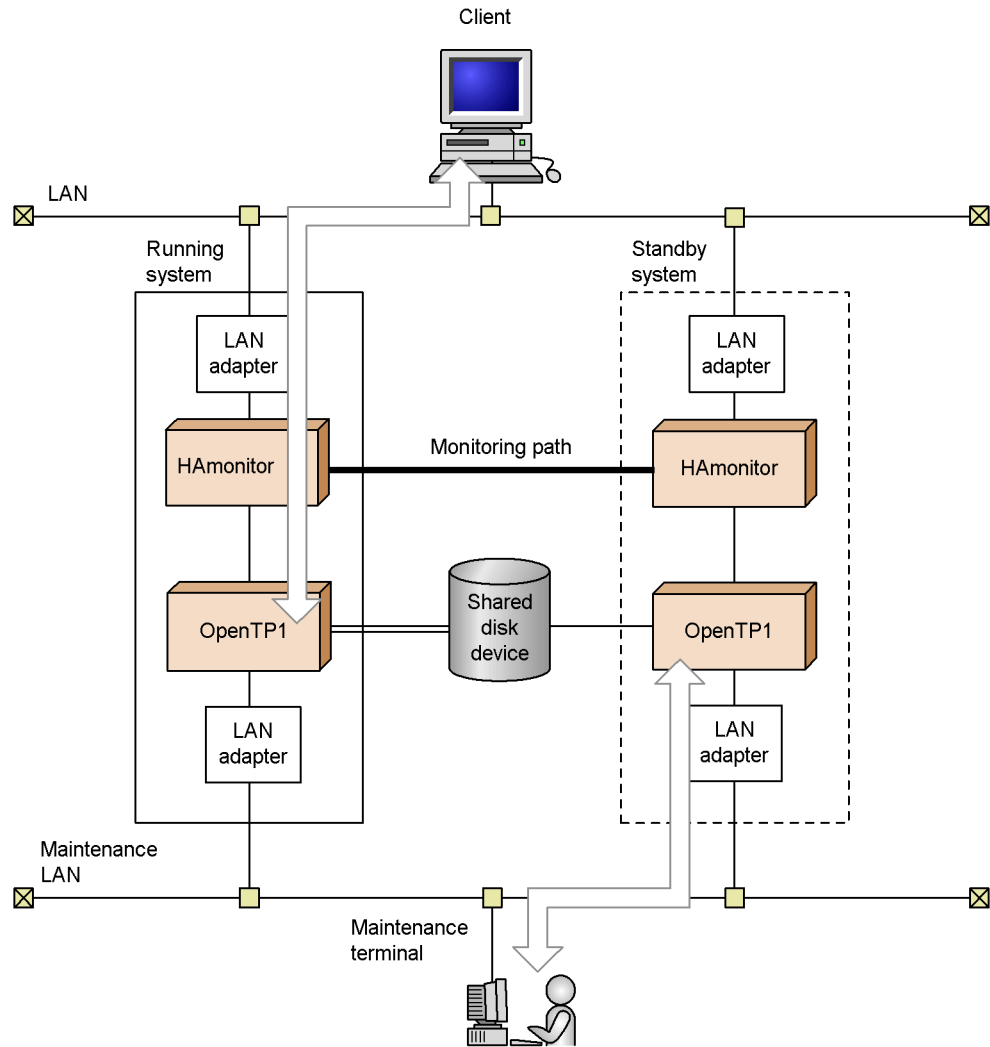
Used for the name service of OpenTP1. The LAN is used for OpenTP1 communication. An additional LAN (called a maintenance LAN) is necessary for maintaining multiple systems. The maintenance LAN is not used for OpenTP1 communication; it is mainly used for remote access to the standby system.

- Monitoring path

An interface for communicating the status of each system. The monitoring path is used by HAmonitor. The monitoring path can be either the *monitoring link* or the *monitoring LAN*.

The figure below shows the configuration when using the System Switchover facility. For the primary and secondary systems to monitor each other, hardware and software additional to that shown in the figure are required. For details about the required resources, see the relevant HAmonitor manuals.

Figure 6-2: Configuration when using the System Switchover facility



### 6.1.3 Procedure for system switching

#### (1) Preparation for system switching

##### (a) Allocate shared disk devices

Specify a character special file to be used for OpenTP1 files. Specify the file to be shared so that both OpenTP1 systems can reference it with the same pathname. Only files created on a character special file can be shared. Contents of ordinary files (e.g.,



files in the directories `$DCDIR/spool/` or `$DCDIR/tmp/`) cannot be inherited.

**(b) Set the IP addresses**

In the `my_host` operand of the system common definition, specify the shared IP address. The same IP address must be specified for both OpenTP1 systems.

**(c) Set the HAmonitor environment**

Set the HAmonitor definitions. For the HAmonitor environment settings, see the relevant HAmonitor manuals.

**(2) HAmonitor definition contents**

In the `server` statement of HAmonitor, specify the environment for OpenTP1 operations. In the `name` operand of the `server` statement, you must specify the full-pathname of the OpenTP1 home directory. For details on other environment settings, see the relevant HAmonitor manuals.

**(3) Ensure that required OpenTP1 definitions match**

The following definitions of the primary system and the secondary system must match:

- OpenTP1 system definitions
- executable files of a user server (UAP)
- executable files of the transaction service (if you re-create the file with the `trnlncrm` command)<sup>#</sup>
- versions of program products in the OpenTP1 system
- environment (user ID, group ID, and environment variables) for the OpenTP1 administrator
- absolute pathname of the OpenTP1 home directory
- setting of OpenTP1 files (character special files)

#

You must also execute the `trnlncrm` command to make sure that the order of resource managers registered in OpenTP1 on the primary system is the same as the order on the secondary system.

OpenTP1 does not check for differences in the definitions of two systems, so if definitions do not match, OpenTP1 operation cannot be guaranteed.

**(a) Set the system definitions**

To use the System Switchover facility:

- Specify `Y` in `ha_conf` in the system configuration definition. If this is not specified, a system switch is not possible.

- The specification of the system startup method in the system environment definition (`mode_conf` operand) differs depending on the reason for starting the standby system, as described below.

When starting the standby system to replace the running system:

Specify `AUTO` or `MANUAL1`.

If `MANUAL2` is specified, an OpenTP1 system which ends abnormally cannot start automatically. To prepare for another system switch, you must use a command to restart the OpenTP1 system.

When starting the standby system just to complete the postprocessing of the running system:

To start the standby system just to complete the postprocessing of the running system, such as deciding undecided transactions and ensuring database integrity, specify `MANUAL2`. This method assumes that, at a minimum, the standby system can perform fall-back operations without any problems. For details about how to have the standby system perform only the postprocessing of the running system, see *6.1.4(1) Starting and terminating OpenTP1*.

- To reduce the restart time of the standby system after a system switch, user servers can be specified to be started before OpenTP1 is started. Specify `Y` in the `user_server_ha` operand in the system environment definition.

#### **(4) Notes on using the System Switchover facility**

##### **(a) LAN components to be used**

When the System Switchover facility is used, the LAN components and communication lines used by OpenTP1 are online in the running system, and offline in the standby system. Whenever the running system switches, the LAN components and communication lines in that system automatically go offline.

If you want to perform maintenance on an offline standby system, you should not use the LAN components and communication lines that are used by OpenTP1. This is because both the running system and the standby system share the same IP address, so there is a possibility of confusion when both systems are performing network-related activities. To perform such maintenance tasks, use a console or prepare a maintenance LAN, one not used in usual system switching.

##### **(b) File system to be used**

When using the System Switchover facility, use the OpenTP1 file system. If an ordinary UNIX file system is used, OpenTP1 might not start.

##### **(c) Using mirrored disks**

Whether the information on a shared disk device can be duplicated using mirrored

disks depends on the operating system and hardware that you use. Use mirrored disks based on the specifications such as those for the mirrored disks and cluster facility of the operating system and hardware.

If a system switchover occurs while writing to two disks using a mirrored disk, inconsistency may occur between the two disks due to a delay in writing. If the written data is not correct in the disk of the system to which the operation should be switched over, OpenTP1 cannot operate normally after a system switchover.

The following OpenTP1 files are essential to operate OpenTP1:

- Status files
- System journal files
- Checkpoint dump files

To use mirrored disks for these OpenTP1 files, make sure that the operating system and hardware allow using mirrored disks for shared disk devices and that the data in the disk devices is correct even when the System Switchover facility is used.

To improve the reliability of files that cannot be written to mirrored disks:

- duplicate the status files and system journal files.
- duplicate the checkpoint dump files or use the multi-generation guarantee facility for checkpoint dump files.

#### 6.1.4 Operating with the System Switchover facility

This subsection describes how to operate OpenTP1 when using the System Switchover facility.

##### (1) *Starting and terminating OpenTP1*

###### (a) **Starting OpenTP1**

By using the OpenTP1 start command `dcstart`, you can start two OpenTP1 systems that use the System Switchover facility. If OpenTP1 is set to start automatically, you do not need to use `dcstart`.

- The system environment definition (i.e., whether or not `manual` is specified) and the type of recent termination determines whether OpenTP1 is started automatically or manually.
- The running or standby system can be specified in the `server` statement that sets the HAmonitor environment.

A system for which `online` is specified in the `initial` operand or for which the `initial` operand is omitted becomes the running system when online operation starts.

A system for which `standby` is specified in the `initial` operand becomes the

standby system when online operation starts.

- Whether OpenTP1 is started normally or restarted is decided in the same way as when the System Switchover facility is not used.

Having the standby system perform the postprocessing of the running system after a system switchover

To have the standby system perform the postprocessing of the running system after a system switchover, execute `dstart -U` to start the standby system.

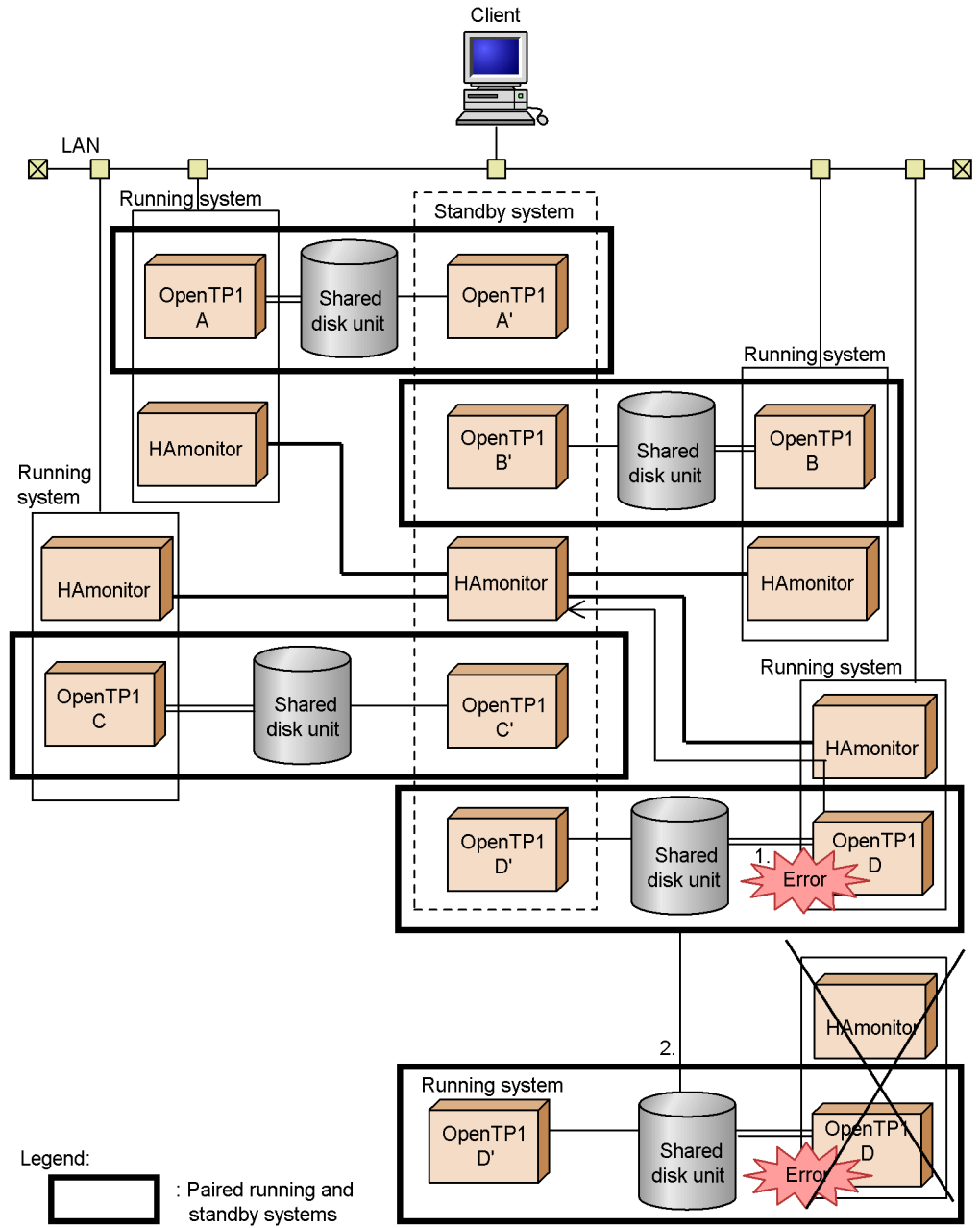
When `dstart -U` is used to start the standby system, the user server does not start at a system switchover. The standby system started in this way can be used for only the postprocessing of the running system, such as deciding undecided transactions and ensuring database integrity. The machine for operating the standby system in this case does not require resources equivalent to those for the running system.

Because the `dstart` command needs to be executed, the standby system when used for postprocessing must be in a state in which it can be started manually.

The following describes an operation when one node is used as a standby system to perform only the postprocessing of multiple OpenTP1 systems.

In this configuration, a standby OpenTP1 system must be set up for each running OpenTP1 system.

Figure 6-3: Operation in which a standby system performs only the postprocessing of a running system after a system switchover



**Explanation:**

1. An error occurs in the OpenTP1 D running system.
2. The OpenTP1 D' standby system starts.

OpenTP1 D' performs postprocessing, such as deciding undecided transactions and ensuring database integrity. Meanwhile, OpenTP1 A, OpenTP1 B, and OpenTP1 C continue online processing in fall-back mode. Because the number of running systems is reduced from 4 to 3, the system performance is degraded by 25%.

**Notes:**

In the following cases, a standby system does not start for the sole purpose of postprocessing:

- When the `mode_conf` operand (system startup method specification) in the system environment definition has been set to `AUTO`, and OpenTP1 terminates abnormally or the OS is started in a situation in which the previous OS termination was not abnormal
- When the `mode_conf` operand (system startup method specification) in the system environment definition has been set to `MANUAL1`, and OpenTP1 terminates abnormally

Even when the operation has been set so that the standby system will perform only the postprocessing of the running system at a system switchover, if one of the conditions above is satisfied, the `dcstart` command is automatically executed. The standby system started by the `dcstart` command without an option waits to take over from the running system.

If you want to start the standby system again for the sole purpose of postprocessing, stop OpenTP1 and then restart it by using the `dcstart -U` command.

**(b) Terminating OpenTP1**

Tables 6-1 and 6-2 describe how to terminate an OpenTP1 system that uses the System Switchover facility.

*Table 6-1: Commands for terminating a running OpenTP1 system*

Termination commands	Execution results for a running OpenTP1 system
<code>dcstop</code> (with no option)	Terminates the running system normally. The standby system also terminates.
<code>dcstop</code> (with <code>-n</code> option)	Forces the running system to terminate normally. The standby system also terminates.

Termination commands	Execution results for a running OpenTP1 system
<code>dcstop</code> (with <code>-a</code> option)	Terminates the running system in the planned termination A mode. The standby system also terminates.
<code>dcstop</code> (with <code>-b</code> option)	Terminates the running system in the planned termination B mode. The standby system also terminates.
<code>dcstop</code> (with <code>-f</code> option)	Terminates the running system forcibly. The standby system also terminates.
<code>monswap</code> (HAMonitor command for a planned system switch)	The systems switch after the running server terminates.
<code>monsbystp</code> (HAMonitor command for terminating the standby system)	Cannot be executed.

Table 6-2: Commands for terminating a standby OpenTP1 system

Termination commands	Execution results for a standby OpenTP1 system
<code>dcstop</code> (with no option)	Cannot be executed.
<code>dcstop</code> (with <code>-n</code> option)	Cannot be executed.
<code>dcstop</code> (with <code>-a</code> option)	Cannot be executed.
<code>dcstop</code> (with <code>-b</code> option)	Cannot be executed.
<code>dcstop</code> (with <code>-f</code> option)	Terminates the standby system forcibly.
<code>monswap</code> (HAMonitor command for a planned system switch)	Cannot be executed.
<code>monsbystp</code> (HAMonitor command for terminating the standby system)	Terminates the standby system.

## (2) Limitations on executing commands

### (a) Note on commands executed offline

When you execute commands offline, you must first terminate both OpenTP1 systems in a system switch configuration, and then execute the commands. You can execute the `dcstart` command if an OpenTP1 system which you want to start is not started. The other OpenTP1 system need not be terminated. A standby system ignores the `dcstart -n` command.

### (b) Note on commands executed online

In a running OpenTP1 system, you can execute commands online. In a standby OpenTP1 system, you cannot execute any commands online except for the `dcstop -f` command (for a forced termination).

To reduce the burden of unloading the running system, system journal files for the standby system can be unloaded. Be careful, however, when executing the `jnlunlfg` command for an unload. For details of unloading the standby system, see the description of the `jnlunlfg` command in the manual *OpenTP1 Operation*.

**(c) Accessing a shared disk device**

Even if you execute a command to perform an operation on a shared file on a shared disk device, OpenTP1 does not write to the file so the file will not be destroyed. Note, however, that if you execute the current system only (with the secondary system usable but not yet activated as an OpenTP1 component), you can write to a shared file from the secondary system and using a command in this case might destroy files. You must terminate both systems before using a command to perform operations on a character special file of OpenTP1.



## 6.2 The Multinode facility

The *Multinode facility* enables efficient management of OpenTP1 systems when OpenTP1 is used in a cluster system or parallel-processing system.

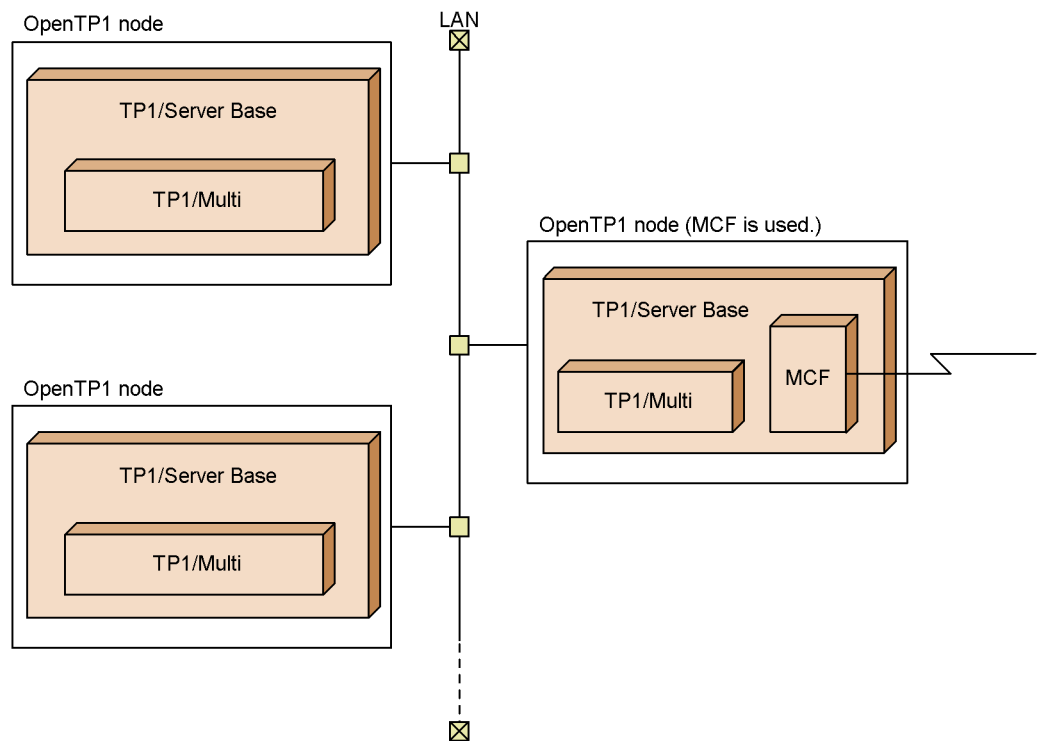
### 6.2.1 Overview of the Multinode facility

In a cluster system or parallel-processing system, multiple server machines connected in a LAN build one large-scale system and operate in parallel. The Multinode facility enables all OpenTP1 systems in a cluster system or parallel-processing system to be operated from one node, thereby decreasing the amount of work required to manage each node in the system.

When using the Multinode facilities, *TP1/Multi* is needed for all the OpenTP1 systems.

The following figure shows the software configuration of an OpenTP1 system that uses the multi-node facility.

*Figure 6-4:* Software configuration of OpenTP1 that uses the Multinode facility



### **(1) Prerequisites for the Multinode facility**

When the Multinode facility is used, the following items are required:

- The TP1/Multi product must be installed on every OpenTP1 node that is to be managed by the Multinode facility.
- In every OpenTP1 node to be managed by the Multinode facility, `multi_node_option=Y` must be specified in the system common definition, and the multinode configuration definition and the multinode physical definition must have been created.

### **(2) Relation between the Multinode facility and other OpenTP1 facilities**

The following facilities are available when OpenTP1 is used in a cluster system or parallel-processing system configuration.

#### **(a) Single OpenTP1 configuration**

In a single OpenTP1 configuration, you can operate ordinary OpenTP1 facilities. UAPs can start in environments set in their OpenTP1 node. Both the following OpenTP1 facilities are available: the Multiserver facility, which starts more than one UAP server process at one time; and the Internode Load-Balancing facility, which assigns processing for a service group to more than one node.

#### **(b) Double OpenTP1 configuration using the System Switchover facility**

In a double OpenTP1 configuration constructed with the System Switchover facility, the set of OpenTP1 instances in a System Switchover configuration is regarded as one node. If the systems switch, the node is considered the same as the former node. Some TP1/Multi facilities (commands, APIs) are unavailable for OpenTP1 instances using the System Switchover facility.

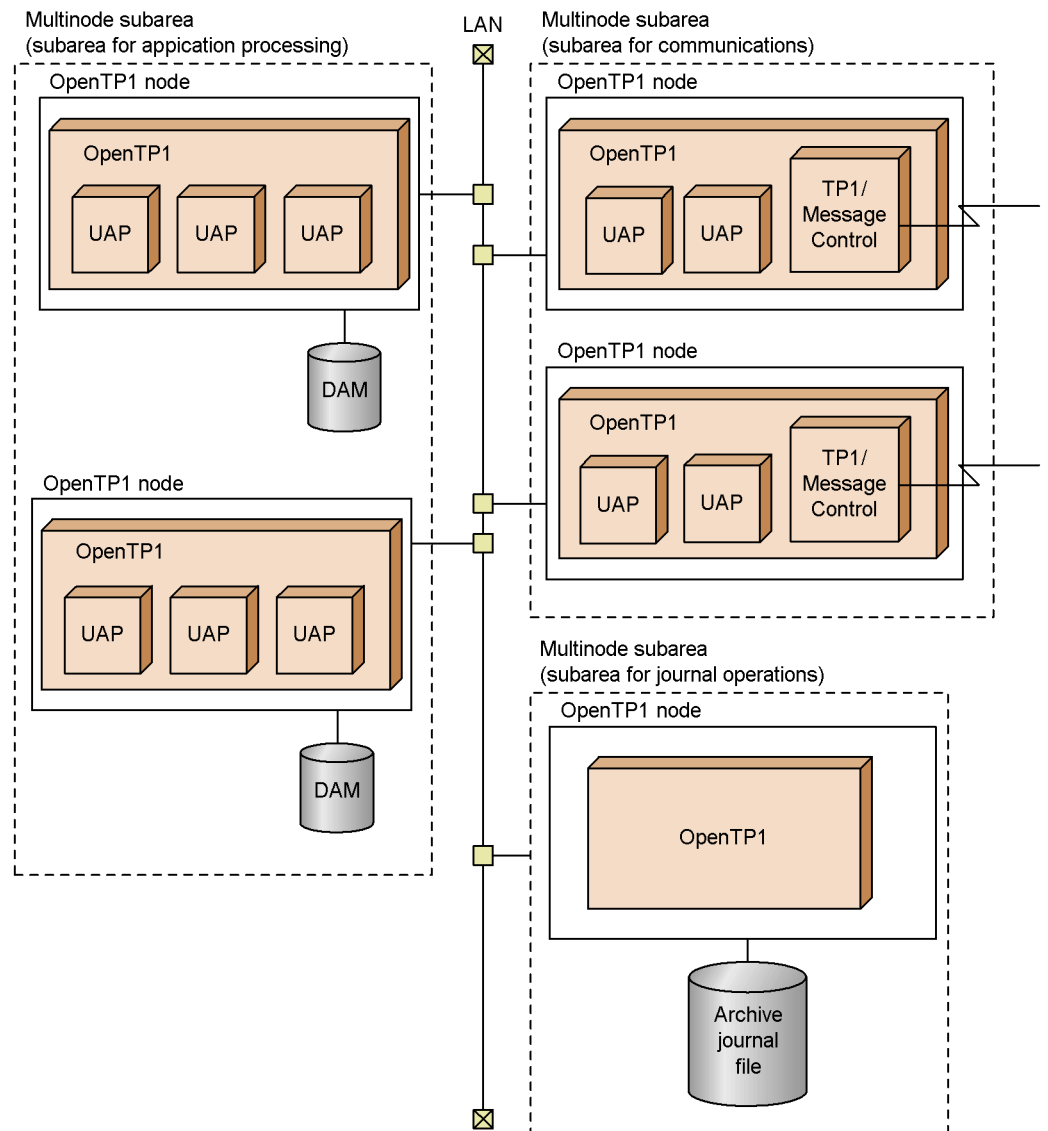
When using the System Switchover facility, a maintenance LAN, one not used for OpenTP1 communication or usual system switching, is required. Specify the host name of the maintenance LAN as the host name in the multinode physical definition.

### **(3) Cluster system or parallel-processing system components**

When OpenTP1 is used in a cluster system or parallel-processing system, the areas that contain multiple systems are managed as described below so that the individual nodes can be managed.

The following figure shows how OpenTP1 is configured when used in a cluster system or parallel-processing system.

Figure 6-5: OpenTP1 configurations in a cluster system or parallel-processing system



**(a) Multinode area**

The set of OpenTP1 instances in a cluster system or parallel-processing system constitute a *multi-node area*. All OpenTP1 nodes to be centrally managed within the system are contained in the multi-node area.

Such a system can have only one multi-node area.

**(b) Multinode subareas**

The multi-node area is logically divided into a set of *multi-node subareas*. OpenTP1 nodes in the multi-node area that perform the same type of processing are grouped into the same subarea. For example, the multi-node area may be divided into the following subareas:

- Subarea for application processing  
A set of OpenTP1 nodes that manage user files (e.g., DAM and TAM files).
- Subarea for WAN communications  
A set of OpenTP1 nodes that communicate in a WAN via TP1/Message Control.
- Subarea for journal operations  
A set of OpenTP1 nodes that use the Global Archive Journal facility to store system journal files from each node into an archive journal file. (For details, see *6.2.3 Global Archive Journal facility*.)

**(c) OpenTP1 nodes**

OpenTP1 systems that constitute a multinode area or a multinode subarea are called *OpenTP1 nodes*. Each OpenTP1 node is distinguished by a node identifier specified in the OpenTP1 system common definition.

**(d) System definitions**

In the multinode configuration definition, you can specify which OpenTP1 nodes belong to each area. As an OpenTP1 node identifier, specify the node identifier you specified in the system common definition.

## 6.2.2 Available operations in the Multinode facility

The following operations can be performed when using the multi-node facility in a cluster system or parallel-processing system configuration. For details on commands, see the manual *OpenTP1 Operation*. For details on functions, see the *OpenTP1 Programming Guide*.

**(1) Starting and terminating OpenTP1 instances in a cluster system or parallel-processing system**

You can enter commands from one OpenTP1 node to start or terminate OpenTP1 instances in a multinode subarea. Use the `dcmstart` command to start OpenTP1 instances in a multinode subarea, and the `dcmstop` command to terminate.

**(2) Collecting status of OpenTP1 nodes**

From one OpenTP1 node you can use the `dcndls` command to monitor the status of OpenTP1 nodes in the multinode area. The status of OpenTP1 nodes can be received

also by issuing the `dc_adm_get_nd_status_xxx()` functions from a UAP.

### **(3) Collecting status of user servers in OpenTP1 nodes**

You can receive the status of user servers in OpenTP1 nodes from one node by issuing the `dc_adm_get_sv_status_xxx()` functions from a UAP.

### **(4) Collecting configuration information of the multinode area**

You can receive information about the system configuration (e.g., all OpenTP1 node identifiers in the multinode area or specified multinode subareas) by issuing the `dc_adm_get_nodeconf_xxx()` functions or the `dc_adm_get_node_id_xxx()` functions from a UAP.

## **6.2.3 Global Archive Journal facility**

As with ordinary OpenTP1 configurations, when OpenTP1 is used in a cluster system or parallel-processing system, you need to unload the system journal file for each OpenTP1 node. Since unloading system journal files from every node can take a lot of time, the OpenTP1 Multinode facility provides the *Global Archive Journal facility* to simplify this procedure. For global archive journals, you can select the kind of journal to be archived by using the `jnl_arc_rec_kind` operand in the system journal definition. For details of the kinds of journals, see *4.2.2 System files: system journal files*.

Archive system journals from each OpenTP1 node into an *archive journal file* on a specific OpenTP1 node. For details on archive journal files, see *4.2.6 System files: archive journal files*.

As with ordinary OpenTP1 configurations, when the Global Archive Journal facility is used, OpenTP1 uses the system journal files for recovering an OpenTP1 node. So using the Global Archive Journal facility does not affect OpenTP1 recovery performance. For details on file operations and error countermeasures, see the manual *OpenTP1 Operation*.

### **(1) Type of nodes that use the Global Archive Journal facility**

In the multinode area, you can make an optional node dedicated to archive journals. Each OpenTP1 node sends system journals to that node in order to archive the journals. A node from which journals are sent to be archived is called an *archive-journal source node*. The node on which the journals are archived is called the *archive-journal target node*.

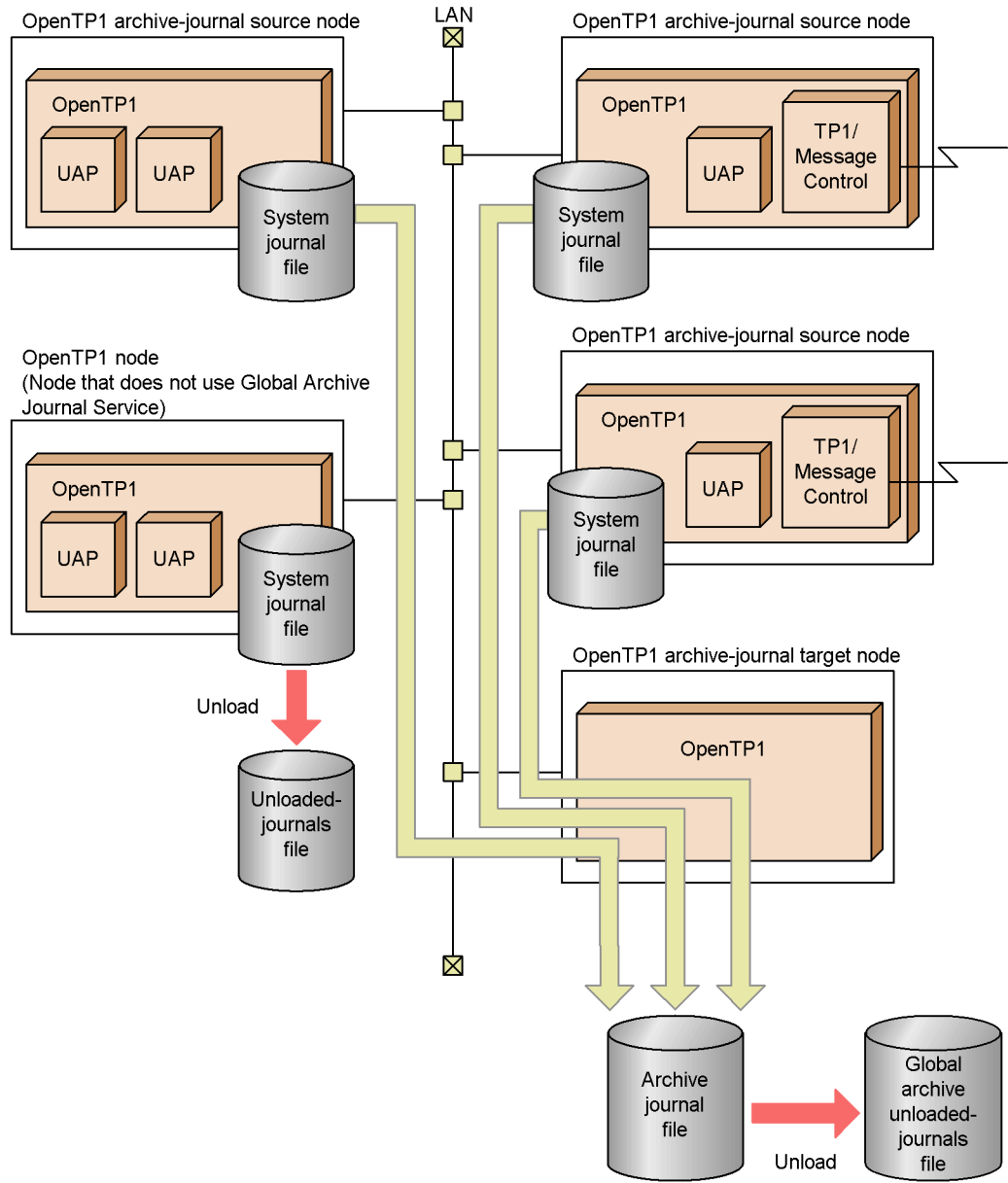
The archive-journal target node is different from other OpenTP1 nodes (archive-journal source node or a node which does not use the Global Archive Journal facility) in the following points:

- user servers cannot be activated on an archive-journal target node
- the archive-journal target node must be first started before any other OpenTP1

node in the multinode area

The following figure illustrates the global archive journal service.

Figure 6-6: Overview of Global Archive Journal facility



**(2) Resource groups**

You can make from 1 to 16 resource groups in an archive-journal target node. A *resource group* is a set of archive journal files classified for the same purpose. For example, you can specify a resource group A that collects journals from one part of the multinode area, and you can specify another resource group B that collects journals from another part of the multinode area.

The global archive journal service uses resource group names to manage archive journal files.

Up to 20 archive journal source nodes can belong to one resource group.

A resource group name is defined in the archive journal service definition as a file name. In the global archive journal service definition, you can specify the number of resource groups to be used in a node on which a journal is archived.

**(3) Definitions in OpenTP1 nodes using the Global Archive Journal facility**

To connect the global archive journal service to a journal service in each node, the following definitions must be specified in nodes that use the Global Archive Journal facility:

- Archive-journal target node

Create the global archive journal service definition and the archive journal service definition.

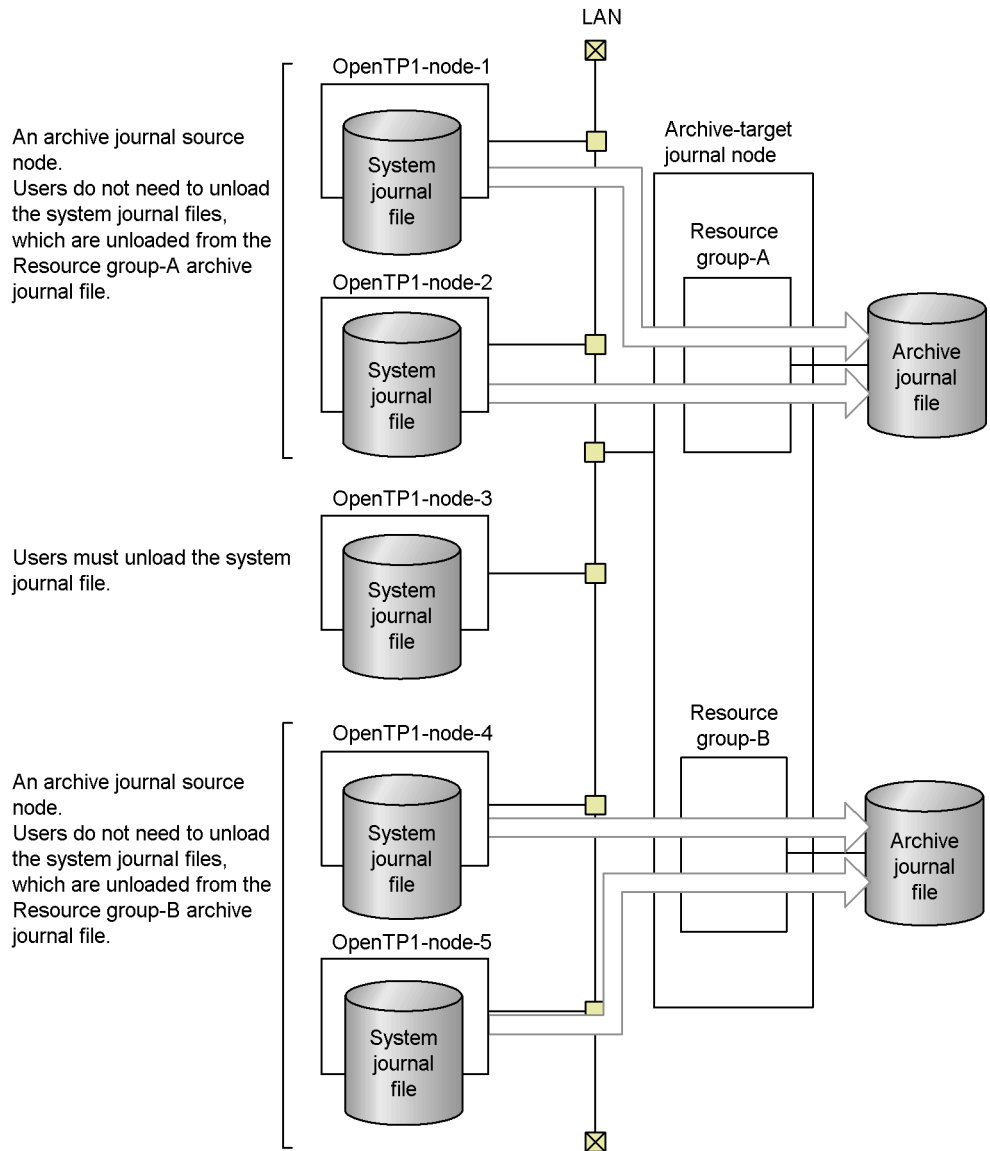
- Archive-journal source node

In the `jnl_arc_name` operand of the system journal service definition, specify the node identifier of the archive-journal target node and resource group names of archive journal files.

You must unload every OpenTP1 node which does not use the Global Archive Journal facility.

The following figure shows the relationship between the global archive journal service and resource groups.

*Figure 6-7: Relation between the global archive journal service and resource groups*



**(4) Unloading archive journal files**

A filegroup that is waiting to be unloaded needs to be copied (unloaded) to a regular file, using an OpenTP1 command. An unloaded archive journal file is called a *global archive unloaded-journals file*.



When one or more archive journal filegroups are not yet unloaded, but no swap destinations are available, the global archive journal service terminates abnormally.

**(a) Journal maintenance of a global archive unloaded-journals file**

As with a system journal file, an unloaded ordinary file can be used for recovering DAM files and editing uptime statistics. Also a user journal can be inherited by an offline program.

A global archive unloaded-journals file contains system journal information about OpenTP1 nodes. You can edit this journal: for example, by editing merged uptime statistics or sorting OpenTP1 node journals by time.

**(5) Status of system journal files**

In an OpenTP1 archive-journal source node, the following statuses of system journal files are added:

- archived/not archived

Indicate whether or not the filegroup has already been sent to and stored in an archive journal file.

If a standby system journal filegroup has been sent to an archive journal file and the filegroup has overwrite-permitted status, the standby system journal filegroup can be swapped in to become the current system journal filegroup regardless of whether or not the archive journal file has been unloaded.

## 6.3 The MultiOpenTP1

The MultiOpenTP1 enables one node to contain two OpenTP1 instances. This configuration is useful when you want to use one OpenTP1 instance for processing and the other for testing applications.

Each OpenTP1 instance can operate independently. In a MultiOpenTP1 configuration, two OpenTP1 instances can share a single node address or use different addresses.

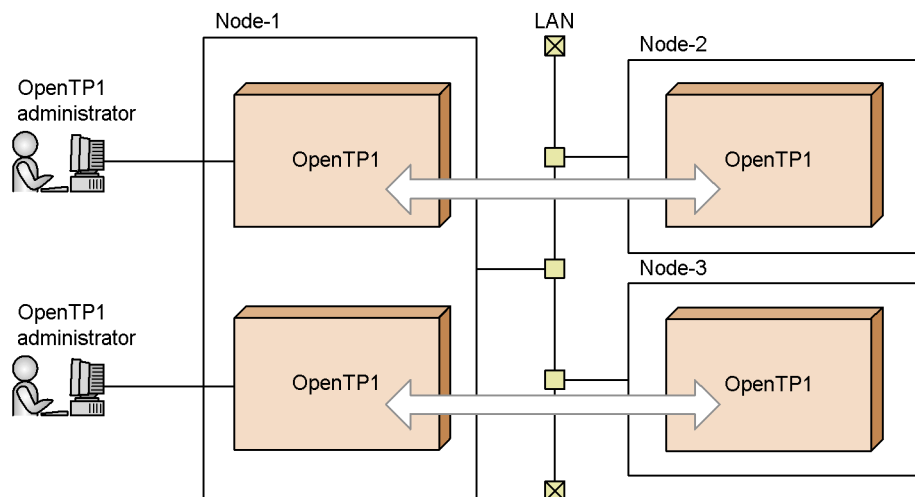
### 6.3.1 MultiOpenTP1 configuration

The two OpenTP1 instances are installed in different OpenTP1 home directories. An OpenTP1 administrator is required for each OpenTP1 instance, although the same person can perform both sets of administration tasks.

In a MultiOpenTP1 configuration, each OpenTP1 instance is identified by an OpenTP1 identifier. The port number for the name service and process service must be unique for each OpenTP1 instance. The OpenTP1 identifier and the port number are specified in the system common definition. For details on setting up a MultiOpenTP1 environment, see the manual *OpenTP1 Operation*.

The following figure illustrates a multiOpenTP1 configuration.

Figure 6-8: A MultiOpenTP1 configuration



#### (1) A sample for distributing MultiOpenTP1 commands

When a command such as `rsh` (a remote shell command) is used to execute a command on another node in a MultiOpenTP1 configuration, OpenTP1 instances on the remote node do not know where the command should be executed. Therefore, a

node name must be specified when a command is executed.

The `delvcmd` command simplifies the way node names are specified in the process described above. The `delvcmd` command is installed in the TP1/Server Base shell file. For details about how to use the `delvcmd` command, see the description about the sample program in the *OpenTP1 Programming Guide*.

---

## 6.4 Multi-homed host configuration

---

A host connecting two or more physical networks with the TCP/IP protocol is called a *multi-homed host*. To communicate with OpenTP1 that runs in such a host, use the IP address of a LAN connected to the host. For normal communication, users need not be aware of the IP address (host name) for that OpenTP1 instance.

If a multi-homed host runs two OpenTP1 instances and uses either of them with the System Switchover facility, a system switch may prevent communication with an OpenTP1 instance on the host. To avoid blocked communication with the OpenTP1 instances on the host, the correspondence between the IP addresses (host names) and OpenTP1 instances must be defined.

### (1) Defining the multi-homed host with system switching

To use either of the OpenTP1 instances in a host with the System Switchover facility, specify the host name in the `my_host` operand in the system common definition and specify the following in the `dcbindht` definition command:

- Name of the host to be used (host name that can be mapped with an IP address in the `/etc/hosts` file or using DNS)
- Name of the network to be used (network name that can be mapped with a network number in the `/etc/networks` file or using NIS)

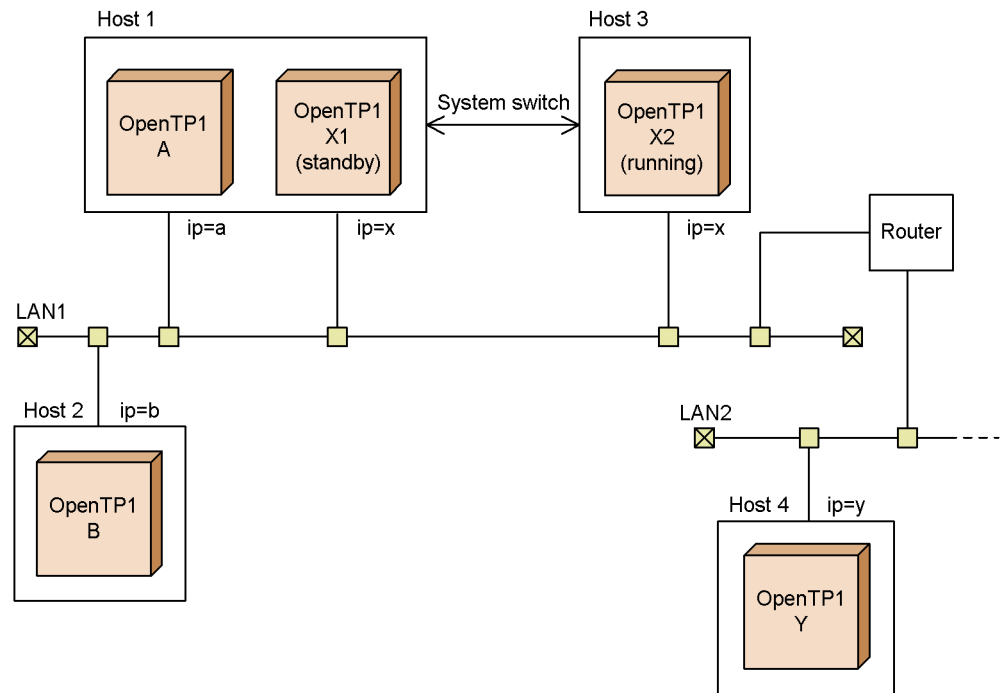
### (2) Example of OpenTP1 instance and IP address correspondence

The following explains a configuration example that requires correspondence between OpenTP1 instances and IP addresses (host names). If one host uses the System Switchover facility in which two OpenTP1 instances run, the `dcbindht` definition command must always be specified.

#### (a) Example of one-to-one system switch configuration

The following figure shows an example of a one-to-one system switch configuration in which two OpenTP1 instances must be associated with a specific IP address (host name).

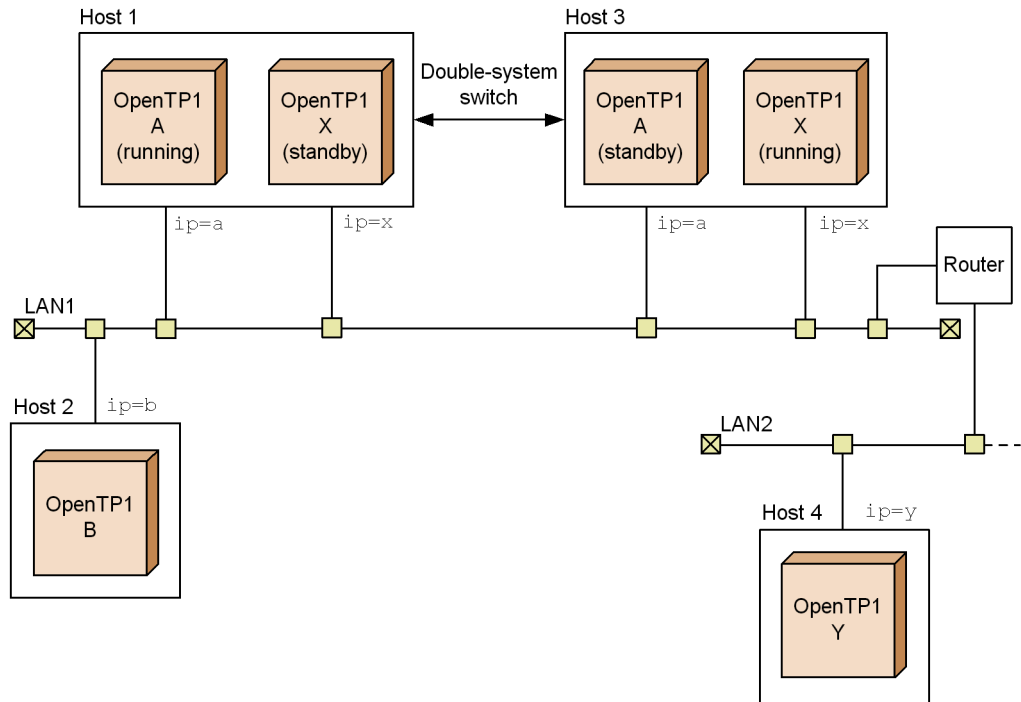
*Figure 6-9: OpenTP1 instances and IP addresses (host names) in a one-to-one system switch*



**(b) Example of double-system switch**

The following figure shows an example of a double-system switch configuration in which two OpenTP1 instances must be associated with a specific IP address (host name).

Figure 6-10: OpenTP1 instances and IP addresses (host names) for a double-system switch



Explanation:

In this figure, the OpenTP1 systems on hosts 1 and 3 are located on both the running and the standby systems.

In such a case, for OpenTP1-A, specify the IP address of `ip=a` in the `-h` option of the `dcbindht` definition command so that the IP address of `ip=b` will not affect communication in the event of system switchover. Similarly, for OpenTP1-X, specify the IP address of `ip=x` in the `-h` option of the `dcbindht` definition command.

## Chapter

---

# 7. System Resources

---

Some knowledge of OpenTP1 process structure and memory structure is necessary to design or manage an online transaction processing system using OpenTP1.

This chapter describes the system resources used by OpenTP1.

- 7.1 OpenTP1 process structure
- 7.2 OpenTP1 memory structure
- 7.3 TCP/IP resources that OpenTP1 uses

## 7.1 OpenTP1 process structure

Table 7-1 lists system service processes.

Table 7-1: System service processes

Name of executable file	Valid number of processes	Service	Relevant service definition	I/O file
prcd	1 <sup>#1</sup>	Process service (superuser process)	Process service definition	None
namd	1	Name service	Name service definition	None
namaudtd	1	Name service	Name service definition	None
scdd	1	Scheduler	Schedule service definition	None
scdmltd	0 or more	Scheduler	Schedule service definition	None
trnd	1	Transaction management service	Transaction service definition	None
trnrvd	1 or more	Transaction recovery service	Transaction service definition	None
trnrmd	1	Resource manager monitoring service	Transaction service definition	None
itvd	1	Interval timer service	Interval service definition	None
stsd	1	Status service	Status service definition	Status file
cpdd	1 or more <sup>#2</sup>	Checkpoint dump service	Journal service definition	Checkpoint dump file
tjld	0 or <sup>#2</sup>	Transaction journal service	None	Transaction recovery journal file
jnld	1	Journal management service	Journal service definition	None
jnlswd	0 or 1 <sup>#2</sup>	Journal file management service	System journal service definition	None
jnliod	0 to 16 <sup>#2, #3</sup>	Journal file I/O service	None	System journal file
	1 to 256 <sup>#4, #5</sup>	Journal file I/O service	None	Archive journal file



Name of executable file	Valid number of processes	Service	Relevant service definition	I/O file
jnlutld	0 or 1 <sup>#6</sup>	Journal utility service	System journal service definition	System journal file
jnlstd	0 or 1	Journal data transmitting service	None	None
jard	1 <sup>#5</sup>	Global archive journal service	Global archive journal service definition	None
jarswd	1 - 16 <sup>#5</sup>	Archive journal file management service	Archive journal service definition	None
jarrvd	1 to 320 <sup>#5, #7</sup>	Journal data archive service	None	None
itvd	1	Interval timer service	Interval timer service definition	None
stsd	1	Status service	Status service definition	Status file
cpdd	$\geq 0$ <sup>#2</sup>	Checkpoint dump service	Journal service definition Checkpoint dump service definition	Checkpoint dump file
tjld	0 or 1 <sup>#2</sup>	Transaction journal service	None	Transaction recovery journal file
qued	0 or 1	Message queue service	Message queue service definition	Message queue file
damd	0 or 1	DAM service	DAM service definition Checkpoint dump service definition	DAM file
tamd	0 or 1	TAM service	TAM service definition	TAM file
tamioid	0 or 1	TAM file I/O service	TAM service definition	TAM file
ismbd	0 or 1	ISAM service	ISAM service definition	ISAM file
istd	0 or 1	IST service	IST service definition	None
logd	1	Log service	Log service definition	Message log file
prfiop <sup>#8</sup>	0, 8, or 9 <sup>#9</sup>	Trace acquisition service for performance verification	System common definition	Trace file

7. System Resources

Name of executable file	Valid number of processes	Service	Relevant service definition	I/O file
cltcond	0 or more	CUP execution service	Client service definition	None
cltd	0 or 1	Client extended service	Client service definition	None
clttrnd	0 or more	Client execution process	Client service definition	None
xatd	0 or 1	XATMI communication service	None	None
xatcd	0 or 1	XATMI service	XATMI communication service definition	None
rmmd	0 or 1	RMM service	RMM service definition	None
admrsvre	1 or more	Partial recovery	Process service definition	None
mcfmngrd	0 or 1	MCF manager	MCF manager definition	Trace file
rmmd	0 or 1	RMM service	RMM service definition	None
mquad	0 or 1	MQA message queuing service (TP1/Message Queue)	MQA service definition	MQA queue file MQ management information file
mapsmgrd	0 or 1	Mapping service	Mapping service definition Mapping service attribute definition	None
mcfcmdsv	0 or 1	MCF online command service	None	None
<i>user-specified-name</i> <sup>#10</sup>	≥ 0	MCF communication service	MCF communication configuration definition MCF application definition	None
mqaiod	0 or more	Queue file input/output service in message queuing (TP1/Message Queue)	MQA service definition	MQA message queue file
mqcdtcp	0 or 1	MQC message queuing service (TP1/Message Queue)	MQA service definition MQC service definition	Trace file
mqaamd	0 or 1	On-line command service for message queuing (TP1/Message Queue)	MQA service definition	MQA queue file

Name of executable file	Valid number of processes	Service	Relevant service definition	I/O file
mqamnd	0 or 1	Completion message monitoring service for message queuing (TP1/Message Queue)	MQA service definition	None
mqtdtcp	0 or more	MQT communication service for message queuing (TP1/Message Queue)	MQT communication configuration definition MQT service definition	MQA queue file Channel management information file Trace file User definition file MQ management information file
mqtmngd	0 or 1	MQT communication manager service for message queuing (TP1/Message Queue)	MQA service definition	Channel management information file
rapclman	1	Remote API facility	RAP-processing client manager service definition	None
raplisnr	1 to 1024	Remote API facility	RAP-processing listener service definition	Trace file
rapsevr	1 to 1024	Remote API facility	RAP-processing listener service definition	Trace file
rtsspp	$\geq 0$	Real-time statistics service	Real-time acquisition item definition	RTS log file
rtssup	$\geq 0$	Real-time statistics service	Real-time acquisition item definition	RTS log file
utod	1	Offline tester	Tester service definition	Trace file

#### Note

*Valid number of processes* is the number of system service processes per node when one OpenTP1 instance is operating within a node.

#### #1

Although there is usually only one process during operation, it might seem that

multiple processes temporarily exist in the following cases. This is because the process starts the OpenTP1 process.

- When OpenTP1 is started by using the `dcstart` command
- When the user server is started by using the `dcsvstart` command
- When the process of a non-resident server is started

#2

The valid number of processes is 0 when `Y` is specified in the `jnl_fileless_option` operand in the system common definition.

#3

The value is determined using the following equation:

$$a \times b$$

*a*: 2 when journals are duplicated; 1 when journals are not duplicated.

*b*: Maximum dispersed files for parallel access when using the parallel access facility for system journal files.

#4

This number is obtained from the following formula:

$$a \times b \times c$$

*a*: 2 if a journal is duplicated; 1 if a journal is not duplicated.

*b*: Maximum number of distributions in the parallel access facility for archive journal files.

*c*: Number of resources specified in the global archive journal service definition.

#5

The process runs in a node which has a global archive journal service.

#6

The valid number of processes is 1 when `Y` is specified in the `jnl_auto_unload` operand in the system journal service definition.

#7

Same as the number of archive-journal source nodes that are being transferred to the archive-journal target node.

#8

This file is a process for obtaining the trace. Although this process is created in extension with `prcd`, the information obtained by this process cannot be

displayed by OpenTP1 `prcls` command. To display the details, execute the operating system's `ps` command.

#9

The valid number of processes is as follows:

- When `N` is specified in the `prf_trace` operand in the system common definition
  - The valid number of processes is 0.
- When `Y` is specified in the `prf_trace` operand in the system common definition
  - When MCF is not used, the valid number of processes is 8.
  - When MCF is used, the valid number of processes is 9.

#10

When TP1/Messaging is used, the file name is fixed to `mcfutcpd` or `mcfupsvd`.

---

## 7.2 OpenTP1 memory structure

---

OpenTP1 uses two types of memory: *local memory* and *shared memory*.

### 7.2.1 Local memory

Local memory is a virtual storage area that can be accessed by one process only. This area is allocated for OpenTP1 process information and buffers. Most of this area is allocated dynamically when a UAP issues an RPC to request a service; however, part of this area is included in the data section for library objects.

### 7.2.2 Shared memory

Shared memory can be shared by more than one process and can be referenced and updated by them. Most of this area is allocated for OpenTP1 control tables and shared buffers. Shared memory is made available via the shared memory function of the operating system and is used to reduce the overhead for communicating between processes.

#### (1) *Shared memory reserved by OpenTP1*

The shared memory used by OpenTP1 is specified in two operands in the system environment definition: Specify the static shared memory size in the `static_shmpool_size` operand, and the dynamic shared memory size in the `dynamic_shmpool_size` operand. OpenTP1 uses the combined total of these two sizes as shared memory. The data to be used during OpenTP1 operation is generally stored in static shared memory, and data that is accessed exclusively on-demand during execution of a service process is placed in dynamic shared memory.

When the buffer area of the shared memory is specified to be shared between the user servers, the amount of shared memory will be saved. For sharing the buffer table, see Section 3.4.4 *Saving shared memory in sharing a buffer area*.

#### (2) *Displaying status of shared memory*

You can use the `dcshmls` command to find out the status of shared memory which is being used during OpenTP1 execution. This command displays the total amount and the usage rate of shared memory. Command options enable you to separately display the status of static shared memory and dynamic shared memory.

#### (3) *Shared memory used by DAM service, TAM service, and IST service*

Part of dynamic shared memory is used to update DAM files. To use dynamic shared memory more efficiently, in the DAM service definition specify how many transaction branches for updating DAM files are to be activated concurrently. When the number of transaction branches is specified, dynamic shared memory for updating a DAM file is allocated at the start of OpenTP1. If more transaction branches than you specified try to update a DAM file concurrently, more dynamic shared memory is newly

allocated at execution of the transaction. The dynamic shared memory which has been allocated temporarily is released when the transaction terminates.

When the DAM service is used, the DAM service allocates shared memory for the buffer area in which DAM blocks are saved. You can specify the memory size of this area in the DAM service definition. This area is separate from the shared memory that OpenTP1 allocates according to what was specified in the system environment definition.

When a TAM service is used, the TAM service allocates shared memory for TAM tables. You can specify the size of this area in the TAM service definition. This area is separate from the shared memory that OpenTP1 allocates according to the system environment definition.

The IST service, when used, allocates shared memory for internode shared tables. You can specify the size of this area in the IST service definition. This area is separate from the shared memory that OpenTP1 allocates according to the system environment definition.

#### **(4) Shared memory used by MCF service**

The MCF service uses both static and dynamic shared memory.

To allocate static shared memory, perform both of the following settings:

- Specify the static shared memory size in the `-p` option of the MCF manager common definition.
- Specify the `static_shmpool_size` operand of the system environment definition, as the total of the following values:  
*value-specified-in-the--p-option-of-the-MCF-manager-common-definition*  
 + *MCF-trace-buffer-size<sup>#</sup>-of-all-MCF-communication-services*  
 + *MCF-trace-buffer-size<sup>#</sup>-of-all-application-startup-services*

#

Calculate the MCF trace buffer size as follows:

*value-specified-in-the-size-operand-of-the-t-option-in-the-mcfttrc-definition-command* × 2

If a shortage of MCF static shared memory occurs, the MCF service automatically allocates an amount of space equal to half the size of the static shared memory (`static_shmpool_size` operand value in the system environment definition) that is specified in the `-p` option in the MCF manager common definition (`mcfmcomn`). This automatic memory allocation can be performed a maximum of 254 times. If a memory shortage occurs thereafter, the MCF service issues the `KFCA10230-E` log message and outputs error information. Note that if a shortage of static common memory is detected during automatic memory allocation before the allocation count reaches 254, the MCF

service immediately issues the KFCA10240-E log message and outputs error information.

Use the `-i` option in the MCF manager common definition (`mcfmcomn`) to specify whether the KFCA10242-I log message is to be issued during additional memory allocation. If you want to be notified of a shortage of static common memory, specify `msg` in the `-i` option to enable message display.

**(5) Shared memory used by real-time statistics service**

In addition to the shared memory defined in the system environment definition, the real-time statistics service uses the shared memory for the RTS service in order to store acquired statistics.

The size of the shared memory for the RTS service is determined by the values specified in the `rts_service_max` and `rts_item_max` operands in the real-time statistics service definition.



---

## 7.3 TCP/IP resources that OpenTP1 uses

---

If you create a large-scale system using RPCs (the `dc_rpc_call` function, the `dc_rpc_call_to` function, and the XATMI interface that uses TCP/IP) of OpenTP1, the number of TCP/IP ports may run short or a communication error may occur.

This section describes how to restrict the number of ports that OpenTP1 uses and how to tune the network environment.

### 7.3.1 Port numbers used in OpenTP1

When you execute the inter-process communications using TCP/IP, you need to establish a connection between processes before transmitting data. Port numbers are assigned to both ends of the connection.

The send port is allocated by the OS within the range of port numbers available for automatic allocation by the OS (short-lived ports). The receive port is also allocated by the OS within the range of port numbers available for automatic allocation by the OS (short-lived ports). Note, however, that some OpenTP1 system services enable you to specify any desired receive port number.

OpenTP1 uses the following port numbers:

- Port numbers that are automatically allocated by the operating system

The operating system determines these numbers and you cannot change them.

- Port numbers that are specified individually

For example, port numbers that are specified by the following operands are applicable.

- `name_port` and `prc_port` operands in the system common definition
- `scd_port` operand in the schedule service definition
- `rap_listen_port` operand in the RAP-processing listener service definition

- Port numbers starting from the value specified in the `rpc_port_base` operand in the system common definition up to the value specified in the `rpc_port_base` operand + `prc_process_count` in the process service definition + 129

The following table shows the system definition operands that enable you to specify a receive port number.

*Table 7-2: System definition operands that enable you to specify a receive port number*

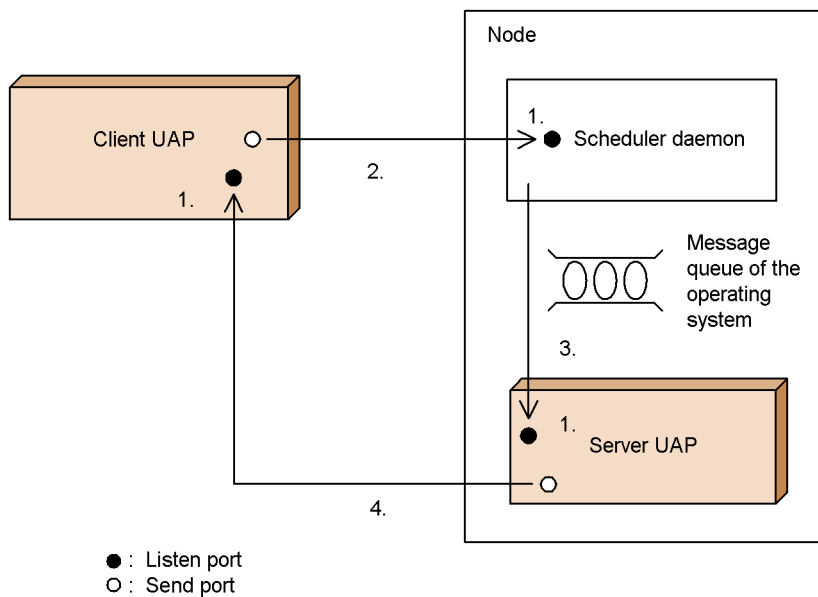
Definition name	Operand	Permitted range	Default value	Applicable system service
System common definition	name_port	5001 to 65535	10000	Name service
	domain_masters_port	5001 to 65535	Range of ports available for automatic allocation	Domain representative schedule service
	prc_port	5001 to 49999	Range of ports available for automatic allocation	Process service
Schedule service definition	scd_port	5001 to 65535	Range of ports available for automatic allocation	Schedule service
Client service definition	clt_port	5001 to 65535	Range of ports available for automatic allocation	Client extended service
	cltcon_port	5001 to 65535	Range of ports available for automatic allocation	Transactional RPC executing process
RAP-processing listener service definition	rap_listen_port	5001 to 65535	Range of ports available for automatic allocation	RAP-processing listener
RAP-processing client manager service definition	rap_client_manager_port	5001 to 65535	Range of ports available for automatic allocation	RAP-processing client manager

For details about the operands, see the manual *OpenTPI System Definition*.

### 7.3.2 How RPCs use ports

The following figure shows how RPCs use ports.

Figure 7-1: How RPCs use ports



1. Each OpenTP1 process (system process or user process) secures a receive port during process initialization.
2. To issue an RPC, the client UAP process secures a send port, establishes a connection with the receive port of the scheduler daemon, and sends the RPC. If a connection is already established, it is used. Each send port is assigned with a different number for each destination process (scheduler daemon of each node).
3. The message queue facility of the operating system is used to contact the server UAP.
4. To return a response to the RPC, the server UAP process secures a send port, establishes a connection with the receive port of the client UAP, and sends the response. If a connection is already established, it is used. Each send port is assigned with a different number for each destination process (client UAP process).

For the subsequent transmission during commitment, the connection that is already established between the client UAP and the server UAP is used.

### 7.3.3 Calculating the number of ports

Table 7-3 describes how to calculate the number of ports used by OpenTP1.

Table 7-3: Formulas for calculating the number of ports that OpenTP1 uses

Purpose	Formula
Accepting TCP/IP communications (receive port)	Number of OpenTP1 system processes + Number of user processes
Communications between system processes (send port)	Number of system processes that communicate with other nodes <sup>#</sup> × Number of other nodes
Sending RPCs (send port)	Number of target nodes which the applicable UAP process sends RPCs to (per UAP process)
Responding to RPCs (send port)	Number of target client processes which the applicable UAP process issues RPCs to (per UAP process)

#

Total number of the following processes:

Number of `namds`+ Number of `istds` (when TP1/Shared Table Access is used)+ Number of `trnrmds`+ Number of `trnrvds` × (Value specified in the `trn_recovery_process_count` operand in the transaction service definition)

### 7.3.4 Restricting the number of ports

If you create a large-scale system using OpenTP1, the number of ports that TCP/IP manages may run short. To prevent this, this appendix describes how to restrict the number of ports that OpenTP1 uses.

To reduce the overhead of establishing connections, OpenTP1 maintains the connections that are established and reuses them for the communications between the same processes. When the number of maintained connections reaches the limit, connections will be disconnected if the processes that established the connections agree with each other. This is *temporary closing*.

The user can specify the following operands in the system common definition, the user service definition, and the user service default definition.

```
set ipc_sockctl_highwater=a,b
set ipc_sockctl_watchtime=length-of-time-to-wait-until-the-sockets-are-reusable
```

*a*

Percentage of sockets at which temporary closing starts

*b*

Percentage of sockets for which temporary closing is not performed

- Specifying the percentage of sockets at which temporary closing starts

OpenTP1 starts temporary closing when the number of file descriptors used for the sockets in the process exceeds the following value:

Value specified in the `max_socket_descriptors` operand  $\times$  (Value a specified in the `ipc_sockctl_highwater` operand/100)

- Specifying the percentage of sockets for which temporary closing is not performed

You can specify the percentage of the number of connections that are not temporarily closed. Calculate the number of connections that are not temporarily closed as follows:

Value specified in the `max_socket_descriptors` operand  $\times$  (Value b specified in the `ipc_sockctl_highwater` operand/100)

- Specifying the length of time to wait until the sockets are reusable

In the `ipc_sockctl_watchtime` operand, specify the length of time (seconds) to wait from the moment the number of file descriptors used for the sockets in the process reaches the value specified in the `max_socket_descriptors` operand until the sockets become reusable due to temporary closing.

In the operation where communications between the same processes are infrequent or in a system that communicates with many remote processes, use temporary closing and release some connections when the number of maintained connections becomes great. This allows you to adjust the number of sockets that are used in a process and reuse the sockets.

When an OpenTP1 process sends data, it secures a send port before it establishes a connection. Since each computer has a limited number of send ports, you need to adjust the number of connections maintained by the UAP processes. Make this adjustment so the total number of send ports in the entire system does not exceed its limit.

If the values specified in these operands are inadequate, the number of sockets that can be used in a process reaches the upper limit. This causes new requests for establishing connections to overwhelm the number of sockets that become reusable by temporary closing. Alternatively, the process may terminate abnormally since the number of ports used in the entire system exceeds the limit of TCP/IP.

### 7.3.5 Temporary closing and user tasks

Only one request for temporary closing can be sent per connection.

Sending the request for temporary closing and receiving the response are handled asynchronously. UAPs can accept new RPCs and send responses to RPCs even though

the responses are not returned to some temporary closing requests for a while. Therefore, temporary closing does not affect user tasks.

### 7.3.6 Monitoring a temporary closing request

The temporary closing request monitoring facility lets you regularly check whether or not a temporary closing request has been received. This facility allows reception of a temporary closing request even when the SPP or MHP of OpenTP1 is waiting to receive a service request.

In temporary closing, a connection established between processes is closed after a necessary interaction between the processes is complete. The interaction consists of one process sending a temporary closing request, and another process receiving the request and returning a response.

That is, when the number of sockets used by a process has reached the maximum<sup>#</sup>, if the process sends a temporary closing request, the connection cannot be closed until the other process receives the request and returns a response.

SPPs and MHPs of OpenTP1 use a `msgrcv` system call while waiting for an incoming service request. Once an SPP or MHP begins to wait as the result of the system call, the connection with the other process cannot be closed because the SPP or MHP cannot receive a temporary closing request from the other process.

The temporary closing request monitoring facility is effective when the number of sockets used by a process has reached the maximum<sup>#</sup> and the process has sent a temporary closing request. In this situation, the facility allows the other process to receive the request and to return a response, even if the other process is waiting for a service request as the result of an `msgrcv` system call.

You can specify the following operands in the user service definition and user service default definition:

`set polling_control_data` = Whether to check the arrival of a socket reuse instruction (temporary closing request)

`set thread_yield_interval` = Interval time to wait for a trigger to receive a socket reuse instruction

#

The maximum number of file descriptors for sockets is specified in the `max_socket_descriptors` operand.

### 7.3.7 Checking an execution status of temporary closing

By obtaining information on the execution status of temporary closing, you can check whether temporary closing executed appropriately while reusing the sockets. Based on this information, you can reduce overhead by specifying optimal values for the `max_socket_descriptors` and `ipc_socket1_highwater` operands according to

the scale and environment of the system.

To obtain information on the execution status of temporary closing, use the `rpcstat` command. This command, when executed, obtains information on the execution status of temporary closing after sending or receiving data for predetermined times.

Table 7-4 lists the information obtained by the `rpcstat` command.

Note

Information on the execution status of temporary closing is not obtained every time data is sent or received. Once every 10 times data is sent or received, this information is obtained.

*Table 7-4:* Information obtainable using the command to check the execution status of temporary closing

Information item	Description
Value specified in the <code>max_socket_descriptions</code> operand	Maximum number of file descriptors for sockets
Value <i>a</i> specified in <code>ipc_socketctl_highwater=a,b</code>	Percentage of sockets at which temporary closing starts
Value <i>b</i> specified in <code>ipc_socketctl_highwater=a,b</code>	Percentage of sockets for which temporary closing is not performed
Number of sockets being used in inter-process communications	Total number of sockets being used in the inter-process communications for reception, in the connecting status and in the connected status
Number of sockets requesting temporary closing	Number of sockets waiting for the process which established the connection to agree to the temporary closing request, because the number of sockets used by the process concerned has exceeded the percentage of sockets at which temporary closing starts
Number of sockets for which temporary closing has been performed	Number of sockets for which temporary closing has completed by agreement with the process that established the connection, because the number of sockets used by the process concerned and the process that established the connection has exceeded the percentage of sockets at which temporary closing starts
Number of sockets for which temporary closing is rejected	Number of sockets for which the process that established the connection does not agree to the temporary closing request
Receive port number	Receive port number used by the applicable process

### 7.3.8 Changes in the size of a resource when the number of sockets increases

OpenTP1 secures the internal management table according to the value specified in the `max_socket_descriptors` operand. Specifying a larger value in the `max_socket_descriptors` operand causes OpenTP1 to secure a larger `malloc` area.

Also, when temporary closing is started, the `malloc` area is secured for managing the sockets that are being temporarily closed. After a response to the temporary closing request is received and the socket is closed, the `malloc` area is released when the socket is reused by another connection.

### 7.3.9 Tuning the network environment

This subsection describes how to prevent the errors that might occur when you create a large-scale system depending on the network environment.

You should control RPCs of OpenTP1 using a smaller value for the timer monitoring value of TCP/IP than the timer value set in the operating system. If you use a network with a low transmission efficiency, such as communication with a remote site via a WAN, a communication failure may occur.

In this case, the user can prevent this problem by specifying the following operands in the system common definition, the user service definition, and the user service default definition.

```
set
ipc_conn_interval=length-of-time-to-wait-until-the-connection-is-established
set ipc_send_interval=interval-of-data-transmission-monitoring
set ipc_send_count=number-of-times-of-data-transmission-monitoring
set ipc_header_rcv_time=length-of-time-to-wait-until-the-communication-control-data-is-received
set
ipc_backlog_count=length-of-queue-storing-connection-establishment-requests
```

- Specifying the length of time to wait until the connection is established

When a connection with the send destination process is not established, OpenTP1 calls the `connect()` system call of TCP/IP. If the connection cannot be established immediately and the socket is in the nonblocking mode, OpenTP1 uses the `select()` system call to monitor the connection until it receives the event of connection establishment completion from TCP/IP.

By specifying the `ipc_conn_interval` operand, you can specify the length of the monitoring time until OpenTP1 receives the event of connection establishment completion from TCP/IP.

- Specifying the interval of data transmission monitoring and the number of times



of data transmission monitoring

After the connection is established, OpenTP1 uses the `write()` system call to write send data into the TCP/IP communication buffer.

The system may not be able to write data into the TCP/IP communication buffer and the `write()` system call may be retried if:

- Packets are frequently lost because of poor transmission quality of the network using a WAN.
- You send data larger than the TCP/IP communication buffer.

By specifying the `ipc_send_interval` operand, you can specify the interval of monitoring data transmission of the `write()` system call.

By specifying the `ipc_send_count` operand, you can specify the number of times data transmission is to be monitored for the `write()` system call.

- Specifying the length of time to wait until the communication control data is received

If you cannot receive the communication control data of OpenTP1 due to an error in the network after you start to receive data, you cannot start the connection setting requests from other processes or start to receive new data for a while.

By specifying the `ipc_header_recv_time` operand, you can check the time until the communication control data is received.

- Specifying the length of queue storing connection establishment requests (listen queue)

OpenTP1 issues the `listen()` system call to a socket which receives a connection establishment request. In the `listen()` system call, `SOMAXCONN` defined by the OS in which OpenTP1 was compiled is specified as the listen queue length. For details about the `SOMAXCONN` value, see the *Release Notes*. If a large number of connection establishment requests are received at the same time, the listen queue may run short and the connection establishment requests may return errors. By specifying the `ipc_backlog_count` operand in the system common definition, you can change the length of the listen queue. The maximum value that can be specified as the listen queue length and the `SOMAXCONN` value depend on the OS. If you want to specify a value greater than the value of `SOMAXCONN` for the OS in the `ipc_backlog_count` operand, check these values in the OS documentation, and make sure that the values you specify are appropriate.

### 7.3.10 Cautions required when using DNS and NIS

OpenTP1 may use DNS or NIS to translate an IP address or obtain an RPC transmission target from a defined host name. Whether these facilities are used

depends on the operating system environment that OpenTP1 uses.

If a failure occurs in DNS or NIS, the transaction processing time may become longer or OpenTP1 or UAP may not be able to start.

To prevent this, design the system taking the following points into account:

- High reliability

Improve the reliability considering the design of an alternative server or resolution within the local server if a failure occurs in the DNS or NIS server.

- Performance

If OpenTP1 uses DNS or NIS, communication may occur when it obtains an address from the host name. Design the system considering this overhead.

---

# Appendixes

---

- A. Communication Protocol Products for Use with TP1/Message Control
- B. Library Functions and Commands
- C. Version Changes
- D. Overview of Remote Procedure Call Processing
- E. Glossary

## **A. Communication Protocol Products for Use with TP1/Message Control**

---

This appendix describes the OpenTP1 products that correspond to the communication protocols. To use these products, TP1/Message Control and TP1/NET/Library are required.

### **A.1 OpenTP1 communication protocol products**

The following describes the OpenTP1 program products that support a specific communication protocol.

#### **(1) Product for networks using the OSI Reference Model**

- TP1/NET/OSI-TP

Enables communication compatible with the OSI TP protocol.

#### **(2) Product for networks using the TCP/IP protocol**

- TP1/NET/TCP/IP

Enables message communication between systems that are connected with the TCP/IP protocol.

#### **(3) Product for GUIs**

- TP1/NET/XMAP3

Enables communication with clients that use XMAP3, and allows clients to use XMAP3.

#### **(4) Product for conventional networks**

- TP1/NET/HDLC

Enables communication compatible with the HDLC procedure.

- TP1/NET/X25

Enables communication that conforms to the X.25 standard (a packet protocol standard for digital transmission).

#### **(5) Product for HNA networks**

- TP1/NET/HNA-NIF

Enables communication compatible with the NIF/HNA protocol.

#### **(6) Product for SNA networks**

- TP1/NET/Secondary Logical Unit - TypeP1

Enables communication compatible with the SNA SLU-TypeP protocol.

This product is used for the primary station.

- TP1/NET/Secondary Logical Unit - TypeP2

Enables communication compatible with the SNA SLU-TypeP protocol.

This product is used for the secondary station.

**(7) Product for User Datagram Protocol**

- TP1/NET/User Datagram Protocol

Enables message communication between systems that are connected with the User Datagram Protocol.

**A.2 Systems connected to protocol products**

Table A-1 shows systems that can be connected to the various OpenTP1 communication-protocol products, and communications lines that can be established between the systems and the OpenTP1 products.

*Table A-1:* Systems that can be connected to OpenTP1 products that correspond to various communication protocols

OpenTP1 product	Protocol	Target system	Communications line
TP1/NET/OSI-TP	OSI TP protocol	VOS3 TMS-4V/SP (OSAS/TP/4VSP) VOS3 XDM/DCCM3 (OSAS/TP/DCCM3) System that uses OSI TP protocol	Leased line (V.24/X.21) Packet network (X.25 (80/84) VC) LAN (CD105/CD10T), other
TP1/NET/OSAS-NIF	NIF/OSI protocol	VOS3 TMS-4V/SP (OSAS/NF/4VSP) VOS3 XDM/DCCM3 (OSAS/UA2/DCCM3)	Leased line (V.24/X.21) Packet network (X.25 (80/84) VC) LAN (CD105/CD10T), other
TP1/NET/User Agent	OSAS/UA protocol	VOS3 TMS-4V/SP (OSAS/UA/4VSP) VOS3 XDM/DCCM3 (OSAS/UA/DCCM3) , other	Leased line (V.24/X.21) Packet network (X.25 (80/84) VC) , LAN (CD105/CD10T), other
TP1/NET/TCP/IP	TCP/IP protocol	Systems providing TCP/IP protocol	LAN (CD105/CD10T), LAN (FDDI), other
TP1/NET/C/S560	C/S560 protocol	CommuniNet	LAN (CD105/CD10T), LAN (FDDI), other
TP1/NET/XMAP3	TCP/IP protocol	XMAP3	LAN (CD105/CD10T) LAN (FDDI), other

A. Communication Protocol Products for Use with TP1/Message Control

<b>OpenTP1 product</b>	<b>Protocol</b>	<b>Target system</b>	<b>Communications line</b>
TP1/NET/HSC	HSC1, HSC2 procedures	HSC1, HSC2systems	HSC1: Leased line HSC2: Telephone network
TP1/NET/HDLC	HDLC procedure	Systems providing HDLC procedure	NRM-slave: Leased line ABM: Leased line ARM: Leased line other
TP1/NET/X25	Packet switching communication (PVC, VC)	System providing X.25 (PVC, VC)	Packet network (X.25 (80/84) PVC, VC), other
TP1/NET/X25-Extended	X25 (VC) communication	X25 (VC) ANSER center system	Packet network (X.25 (80/84) VC) , other
TP1/NET/NCSB	Joint bank system procedures	NCSBsystem	NCSB: Leased line
TP1/NET/HNA-NIF	HNA/NIF protocol (secondary station)	Systems providing HNA/NIF protocol	Leased line (HDLC-NRM secondary station) Packet network (X.25 (80/84) VC) Packet network (X.25 (80/84) PVC), other
TP1/NET/Secondary Logical Unit - TypeP1	SNA protocol (primary station)	Systems providing SNA (SLU-TypeP) protocol	Leased line (HDLC-NRM primary station) Packet network (X.25 (80/84) PVC, VC), other
TP1/NET/Secondary Logical Unit - TypeP2	SNA protocol (secondary station)	Systems providing SNA (SLU-TypeP) protocol	Leased line (HDLC-NRM secondary station) Packet network (X.25 (80/84) VC) , Packet network (X.25 (80/84) PVC), other
TP1/NET/User Datagram Protocol	User Datagram Protocol	User Datagram Protocol system	LAN (CD105/CD10T), LAN (FDDI), other

## B. Library Functions and Commands

The library functions that can be used in OpenTP1 are listed below. For details about library functions related to the message queuing facility (TP1/Message Queue), see the *TP1/Message Queue User's Guide*.

Table B-1: OpenTP1 library functions

	Purpose	Library function name	
		C library	COBOL-UAP creation program
Remote procedure call	Starts a UAP	dc_rpc_open	CBLDCRPC('OPEN')
	Starts an SPP service	dc_rpc_mainloop	CBLDCRSV('MAINLOOP')
	Requests a remote service	dc_rpc_call	CBLDCRPC('CALL')
	Invokes a remote service with a communication destination specified <sup>#1</sup>	dc_rpc_call_to	None
	Receives asynchronous processing results	dc_rpc_poll_any_replies	CBLDCRPC('POLLANYR')
	Obtains the descriptor of an asynchronous-response RPC request where an error occurred	dc_rpc_get_error_descriptor	CBLDCRPC('GETERDES')
	Stops receiving processing results	dc_rpc_discard_further_replies	CBLDCRPC('DISCARDF')
	Refuses to receive the result of specific processing	dc_rpc_discard_specific_reply	CBLDCRPC('DISCARDS')
	Retries a service function	dc_rpc_service_retry	CBLDCRPC('SVRETRY')
	Assigns schedule priorities to requests for a service	dc_rpc_set_service_prio	CBLDCRPC('SETSVPRI')
	References schedule priority of requests for a service	dc_rpc_get_service_prio	CBLDCRPC('GETSVPRI')
	References response-wait time for a service	dc_rpc_get_watch_time	CBLDCRPC('GETWATCH')

B. Library Functions and Commands

Purpose		Library function name	
		C library	COBOL-UAP creation program
	Renews response-wait time for a service	dc_rpc_set_watch_time	CBLDRPC('SETWATCH')
	Obtains a node address of a client UAP	dc_rpc_get_callers_address	CBLDRPC('GETCLADR')
	Obtains a node address of a gateway	dc_rpc_get_gateway_address	CBLDRPC('GETGWADR')
	Sends data to a CUP	dc_rpc_cltsend	CBLDRPC('CLTSEND')
	Terminates a UAP	dc_rpc_close	CBLDRPC('CLOSE')
Remote API facility	Establishes a connection with a rap listener	dc_rap_connect	CBLDCRAP('CONNECT') CBLDCRAP('CONNECTX')
	Disconnects rap listeners	dc_rap_disconnect	CBLDCRAP('DISCNCT')
Transaction Control	Starts a transaction	dc_trn_begin	CBLDCTRN('BEGIN')
	Executes a chained commit operation	dc_trn_chained_commit	CBLDCTRN('C-COMMIT')
	Executes a chained rollback operation	dc_trn_chained_rollback	CBLDCTRN('C-ROLL')
	Executes an unchained commit operation	dc_trn_unchained_commit	CBLDCTRN('U-COMMIT')
	Executes an unchained rollback operation	dc_trn_unchained_rollback	CBLDCTRN('U-ROLL')
	Displays present transaction information	dc_trn_info	CBLDCTRN('INFO')
System Management	Executes a command	dc_adm_call_command	CBLDCADM('COMMAND')
	Reports completion of user-server startup	dc_adm_complete	CBLDCADM('COMPLETE')
	Reports user server status	dc_adm_status	CBLDCADM('STATUS')
Audit log output	Outputs an audit log	dc_log_audit_print	CBLDCADT('PRINT')
Message Log Management	Outputs the message log	dc_logprint	CBLDCLOG('PRINT')



Purpose		Library function name	
		C library	COBOL-UAP creation program
Obtaining User Journal	Obtains user journals	dc_jnl_ujput	CBLDJUNL('UJPUT')
Journal data editing <sup>#2</sup>	Closes a jnlrput output file	None	CBLDJUP('CLOSERPT')
	Opens a jnlrput output file	None	CBLDJUP('OPENRPT')
	Inputs journal data from a jnlrput output file	None	CBLDJUP('RDGETRPT')
Message control facility	Opens an MCF environment	dc_mcf_open	CBLDJUP('OPEN')
	Starts the MCF service	dc_mcf_mainloop	CBLDJUP('MAINLOOP')
	Receives messages	dc_mcf_receive	CBLDJUP('RECEIVE')
	Sends a response message	dc_mcf_reply	CBLDJUP('REPLY')
	Sends a message	dc_mcf_send	CBLDJUP('SEND')
	Resends a message	dc_mcf_resend	CBLDJUP('RESEND')
	Receives a synchronous message	dc_mcf_recvsync	CBLDJUP('RECVSYNC')
	Sends a synchronous message	dc_mcf_sendsync	CBLDJUP('SENDSYNC')
	Sends and receives synchronous messages	dc_mcf_sendrecv	CBLDJUP('SENDRECV')
	Receives temporary stored data	dc_mcf_tempget	CBLDJUP('TEMPGET')
	Modifies temporary stored data	dc_mcf_tempput	CBLDJUP('TEMPPUT')
	Stops continuous inquiry-response	dc_mcf_contend	CBLDJUP('CONTEND')
	Starts an application program	dc_mcf_execap	CBLDJUP('EXECAP')
Reports the application information	dc_mcf_ap_info	CBLDJUP('APINFO')	

B. Library Functions and Commands

Purpose		Library function name	
		C library	COBOL-UAP creation program
	Reports the application information to user exit routines	dc_mcf_ap_info_uoc	None
	Sets user timer monitoring	dc_mcf_timer_set	CBLDCMCF('TIMERSET')
	Cancels user timer monitoring	dc_mcf_timer_cancel	CBLDCMCF('TIMERCAN')
	Executes a commit operation on an MHP	dc_mcf_commit	CBLDCMCF('COMMIT')
	Executes a rollback operation on an MHP	dc_mcf_rollback	CBLDCMCF('ROLLBACK')
	Closes an MCF environment	dc_mcf_close	CBLDCMCF('CLOSE')
	Acquires the status of the MCF communication service	dc_mcf_tlscom	CBLDCMCF('TLSCOM')
	Acquires the connection status	dc_mcf_tlscon	CBLDCMCF('TLCN')
	Establishes connection	dc_mcf_tactcn	CBLDCMCF('TACTCN')
	Releases connection	dc_mcf_tdctcn	CBLDCMCF('TDCTCN')
	Acquires the acceptance status for a server-type connection establishment request	dc_mcf_tlsln	CBLDCMCF('TSLN')
	Starts accepting a server-type connection establishment request	dc_mcf_tonln	CBLDCMCF('TONLN')
	Stops accepting a server-type connection establishment request	dc_mcf_tofln	CBLDCMCF('TOFLN')
	Deletes application timer start requests	dc_mcf_adltap	CBLDCMCF('ADLTAP')
	Acquires the status of a logical terminal	dc_mcf_tlsle	CBLDCMCF('TSLLE')
	Shuts down a logical terminal	dc_mcf_tdctle	CBLDCMCF('TDCTLE')

Purpose		Library function name	
		C library	COBOL-UAP creation program
	Releases a logical terminal from shutdown status	dc_mcf_tactle	CBLDCMCF('TACTLE')
	Deletes a logical terminal's output queue	dc_mcf_tdlqle	CBLDCMCF('TDLQLE')
DAM File Service	Opens a logical file	dc_dam_open	CBLDCDAM('DCDAMSV', 'OPEN')
	Reads a logical file block	dc_dam_read	CBLDCDAM('DCDAMSV', 'READ')
	Rewrites a logical file block	dc_dam_rewrite	CBLDCDAM('DCDAMSV', 'REWT')
	Writes a logical file block	dc_dam_write	CBLDCDAM('DCDAMSV', 'WRIT')
	Closes a logical file	dc_dam_close	CBLDCDAM('DCDAMSV', 'CLOS')
	Shuts down a logical file	dc_dam_hold	CBLDCDAM('DCDAMSV', 'HOLD')
	Releases a logical file shutdown	dc_dam_release	CBLDCDAM('DCDAMSV', 'RLES')
	References logical file status	dc_dam_status	CBLDCDAM('DCDAMSV', 'STAT')
	Starts the use of an unrecoverable DAM file	dc_dam_start	CBLDCDAM('DCDAMSV', 'STRT')
	Terminates the use of an unrecoverable DAM file	dc_dam_end	CBLDCDAM('DCDAMSV', 'END')
	Assigns a physical file	dc_dam_create	CBLDCDMB('DCDAMINT', 'CRAT')
	Opens a physical file	dc_dam_iopen	CBLDCDMB('DCDAMINT', 'OPEN')
	Reads a physical file block	dc_dam_get	CBLDCDMB('DCDAMINT', 'GET')
	Writes a physical file block	dc_dam_put	CBLDCDMB('DCDAMINT', 'PUT')
	Searches for a physical file block	dc_dam_bseek	CBLDCDMB('DCDAMINT', 'BSEK')
	Directly inputs a block from a physical file	dc_dam_dget	CBLDCDMB('DCDAMINT', 'DGET')
	Directly outputs a block to a physical file	dc_dam_dput	CBLDCDMB('DCDAMINT', 'DPUT')
Closes a physical file block	dc_dam_iclose	CBLDCDMB('DCDAMINT', 'CLOS')	

B. Library Functions and Commands

Purpose		Library function name	
		C library	COBOL-UAP creation program
TAM File Service	Obtains TAM table status <sup>#1</sup>	dc_tam_open	None
	Reads a TAM table record	dc_tam_read	CBLDCTAM('FxxR')('FxxU')
	Overwrites a TAM table record	dc_tam_rewrite	CBLDCTAM('MFY')('MFYS')('STR')
	Modifies or adds a TAM table record	dc_tam_write	CBLDCTAM('MFY')('MFYS')('STR')
	Deletes a TAM table record	dc_tam_delete	CBLDCTAM('ERS')('ERSR')
	Cancels reading a TAM table record <sup>#1</sup>	dc_tam_read_cancel	None
	Obtains TAM table status	dc_tam_get_inf	CBLDCTAM('GST')
	Obtains TAM table information	dc_tam_status	CBLDCTAM('INFO')
	Closes a TAM table <sup>#1</sup>	dc_tam_close	None
IST Service	Opens an internode shared table	dc_ist_open	CBLDCIST('DCISTSVC','OPEN')
	Inputs a record from the internode shared table	dc_ist_read	CBLDCIST('DCISTSVC','READ')
	Inputs an internode shared table record	dc_ist_write	CBLDCIST('DCISTSVC','WRIT')
	Deletes an internode shared table record	dc_ist_close	CBLDCIST('DCISTSVC','CLOS')
Resource Locking	Locks resources	dc_lck_get	CBLDCLCK('GET')
	Releases all resource locks	dc_lck_release_all	CBLDCLCK('RELALL')
	Releases a specified resource lock	dc_lck_release_by_name	CBLDCLCK('RELNAME')
XATMI Interface	Calls a request/response service and receives its response	tpcall()	TPCALL

Purpose		Library function name	
		C library	COBOL-UAP creation program
	Calls a request/response service	tpacall()	TPACALL
	Receives asynchronous responses from a request/response service	tpgetrply()	TPGETRPLY
	Cancels a request/response service	tpcancel()	TPCANCEL
	Establishes a connection with a conversational service	tpconnect()	TPCONNECT
	Disconnects conversational services	tpdiscon()	TPDISCON
	Receives a message from conversational services	tprecv()	TPRECV
	Sends a messages to conversational services	tpsend()	TPSEND
	Allocates a typed buffer	tpalloc()	None <sup>#3</sup>
	Releases a typed buffer	tpfree()	None <sup>#3</sup>
	Changes a typed buffer size	tprealloc()	None <sup>#3</sup>
	Obtains typed buffer information	tptypes()	None <sup>#3</sup>
	Propagates a service name	tpadvertise()	TPADVERTISE
	Cancels propagating a service name	tpunadvertise()	TPUNADVERTISE
	A template for service functions	tpservice()	TPSVCSTART
	Returns from a service function	tpreturn()	TPRETURN
TX interface	Starts a transaction	tx_begin()	TXBEGIN
	Executes a commit operation	tx_commit()	TXCOMMIT
	Returns information about the present transaction	tx_info()	TXINFORM

B. Library Functions and Commands

Purpose		Library function name	
		C library	COBOL-UAP creation program
	Opens a resource managers group	tx_open()	TXOPEN
	Executes a rollback operation	tx_rollback()	TXROLLBACK
	Closes a resource managers group	tx_close()	TXCLOSE
	Sets commit_return	tx_set_commit_return()	TXSETCOMMITRET
	Sets transaction_control	tx_set_transaction_control()	TXSETTRANCTL
	Sets transaction_timeout	tx_set_transaction_timeout()	TXSETTIMEOUT
Multinode Facility <sup>#1</sup>	Starts obtaining OpenTP1 node status	dc_adm_get_nd_status_begin	None
	Obtains OpenTP1 node status	dc_adm_get_nd_status_next	None
	Obtains specified OpenTP1 node status	dc_adm_get_nd_status	None
	Stops obtaining OpenTP1 node status	dc_adm_get_nd_status_done	None
	Starts obtaining a node identifier	dc_adm_get_nodeconf_begin	None
	Obtains a node identifier	dc_adm_get_nodeconf_next	None
	Stops obtaining a node identifier	dc_adm_get_nodeconf_done	None
	Obtains the local node identifier	dc_adm_get_node_id	None
	Starts obtaining user server status	dc_adm_get_sv_status_begin	None
	Obtains user server status	dc_adm_get_sv_status_next	None
	Obtains specified user server status	dc_adm_get_sv_status	None

Purpose		Library function name	
		C library	COBOL-UAP creation program
	Exits obtaining user server status	dc_adm_get_sv_status_done	None
Online Tester Control	Reports user server test status	dc_uto_test_statuses	CBLDCUTO('T-STATUS')
Performance Verification Trace	Acquires user-specific performance verification traces	dc_prf_utrace_put	CBLDCPRF('PRFPUT')
	Acquires the sequential number for an acquired performance verification trace	dc_prf_get_trace_num	CBLDCPRF('PRFGETN')
Real-time statistics service	Acquires real-time statistics on a specific section	dc_rts_utrace_put	CBLDCRTS('RTSPUT')

#1

Cannot be used in a program used for creating a COBOL UAP.

#2

Journal data cannot be edited using an API written in C.

#3

No COBOL API exists for the XATMI interface.

OpenTP1 commands are listed below. For details about commands related to the message queuing facility (TP1/Message Queue), see the *TP1/Message Queue User's Guide*.

Table B-2: OpenTP1 commands

Purpose		Command name	Access right
System management	Registers or deletes OpenTP1 in the OS	dcsetup	Superuser
	Redefines and restarts process service	dcreset	OpenTP1 administrator
	Sets up the environment for OpenTP1 startup	dcmakeup	OpenTP1 administrator
	Starts an OpenTP1 system	dcstart	OpenTP1 administrator

B. Library Functions and Commands

Purpose	Command name	Access right
Terminates an OpenTP1 system (Note: To execute the <code>dcstop</code> command from a UAP, execute the command in the background.)	<code>dcstop</code>	OpenTP1 administrator <sup>#</sup>
Outputs system statistics	<code>dcstats</code>	OpenTP1 administrator
Starts nodes in a cluster system or parallel-processing system	<code>dcmstart</code>	OpenTP1 administrator
Stops nodes in a cluster system or parallel-processing system	<code>dcmstop</code>	OpenTP1 administrator
Executes an OpenTP1 command from a scenario template	<code>dcjcmdex</code>	OpenTP1 administrator
Specifies operands in the system definition	<code>dcjchconf</code>	OpenTP1 administrator
Updates the domain definition file	<code>dcjnamch</code>	OpenTP1 administrator
Displays OpenTP1 node status	<code>dcndls</code>	General user
Displays shared memory use	<code>dcshmls</code>	General user
Displays the execution status of temporary closing	<code>rpcstat</code>	General user
Redirects the standard output and standard error output of OpenTP1	<code>prctee</code>	OpenTP1 administrator
Terminates and restarts the <code>prctee</code> process	<code>prctctrl</code>	Superuser
Acquires maintenance information	<code>dcrasget</code>	OpenTP1 administrator
Edits and outputs real-time system statistics to the standard output	<code>dcreport</code>	OpenTP1 administrator
Deletes troubleshooting information	<code>dccspool</code>	OpenTP1 administrator
Checks system definitions	<code>dcdefchk</code>	OpenTP1 administrator
Displays product information	<code>dcplist</code>	OpenTP1 administrator



	<b>Purpose</b>	<b>Command name</b>	<b>Access right</b>
Server management	Starts a server	dcsvstart	OpenTP1 administrator
	Terminates a server	dcsvstop	OpenTP1 administrator
	Displays server status	prcls	General user
	Displays a search pathname for a user server and for a command invoked from the user server	prcpathls	General user
	Modifies a search pathname for a user server and for a command invoked from the user server	prcpath	OpenTP1 administrator
	Terminates OpenTP1 process forcibly	prckill	OpenTP1 administrator
Schedule management	Displays schedule status	scdls	General user
	Shuts down scheduling	scdhold	OpenTP1 administrator
	Releases scheduling shutdown	scdrles	OpenTP1 administrator
	Changes the number of processes	scdchprc	OpenTP1 administrator
	Stops and restarts a process	scdrsprc	OpenTP1 administrator
Transaction management	Displays transaction status	trnls	General user
	Executes a commit operation on a transaction	trncmt	OpenTP1 administrator
	Executes a rollback operation on a transaction	trnrbk	OpenTP1 administrator
	Forcibly terminates a transaction	trnfgt	OpenTP1 administrator
	Starts and stops obtaining transaction statistics	trnstics	OpenTP1 administrator
	Deletes the undecided transaction information file	trndlinf	OpenTP1 administrator

B. Library Functions and Commands

	<b>Purpose</b>	<b>Command name</b>	<b>Access right</b>
	Displays the undecided transaction information of OSI TP communication	tptrnls	General user
XA resource management	Displays the XAR event trace information	xarevtr	General user
	Displays the status of an XAR file	xarfills	General user
	Changes the status of an XAR transaction	xarforce	OpenTP1 administrator
	Shuts down the XAR resource service	xarhold	OpenTP1 administrator
	Creates an XAR file	xarinit	OpenTP1 administrator
	Displays the XAR transaction information	xarls	General user
	Releases the XA resource service from the shutdown status	xarrles	OpenTP1 administrator
	Deletes an XAR file	xarm	OpenTP1 administrator
Lock management	Displays lock information	lckls	General user
	Displays pool information of tables used for locks	lckpool	General user
	Deletes a deadlock information file and timeout information file	lckrminf	OpenTP1 administrator
Name management	Checks the startup of OpenTP1 and deletes cache	namalivechk	OpenTP1 administrator
	Registers or deletes the domain-alternate schedule service	namdomainsetup	Superuser
	Checks the domain configuration (by using the system common definition)	namndchg	OpenTP1 administrator
	Checks the domain configuration (by using the domain definition file)	namchgfl	OpenTP1 administrator
	Forcibly invalidates the startup notification information	namunavl	OpenTP1 administrator
	Displays OpenTP1 server information	namsvinf	OpenTP1 administrator

	<b>Purpose</b>	<b>Command name</b>	<b>Access right</b>
	Operates the RPC control list	namblad	OpenTP1 administrator
Message log management	Displays contents of message log	logcat	General user
	Switches real-time output function of message log files	logcon	OpenTP1 administrator
Audit log management	Sets the environment for audit logging	dcauditsetup	Superuser
OpenTP1 file management	Initializes OpenTP1 file system	filmkfs	OpenTP1 administrator
	Displays OpenTP1 file system status	filstatfs	General user
	Displays contents of OpenTP1 file system	fills	General user
	Backs up OpenTP1 file system	filbkup	OpenTP1 administrator
	Recovers OpenTP1 file system	filrstr	OpenTP1 administrator
	Changes OpenTP1 file group	filchgrp	OpenTP1 administrator
	Changes access permission for an OpenTP1 file	filchmod	OpenTP1 administrator
	Changes OpenTP1 file owner	filchown	OpenTP1 administrator
Status file management	Creates and initializes a status file	stsinit	OpenTP1 administrator
	Displays status file status	stsls	General user
	Displays contents of a status file	stsfills	General user
	Opens a status file	stsoopen	OpenTP1 administrator
	Closes a status file	stsclose	OpenTP1 administrator
	Deletes a status file	stsrn	OpenTP1 administrator
	Swaps status files	stsswap	OpenTP1 administrator

	<b>Purpose</b>	<b>Command name</b>	<b>Access right</b>
Journal file management	Initializes journal files	jnlinit	OpenTP1 administrator
	Displays journal file information	jnlis	General user
	Displays journal information read during a restart	jnlrinfg	OpenTP1 administrator
	Opens journal files	jnlpnfg	OpenTP1 administrator
	Closes journal files	jnlclsfg	OpenTP1 administrator
	Allocates physical journal files	jnladdpf	OpenTP1 administrator
	Deletes physical journal files	jnldehpf	OpenTP1 administrator
	Swaps journal files	jnlswpfg	OpenTP1 administrator
	Deletes journal files	jnlrm	OpenTP1 administrator
	Modifies journal file status	jnlchgfg	OpenTP1 administrator
	Unloads journal files	jnlunlfg	OpenTP1 administrator
	Controls the automatic unloading facility	jnlautnl	OpenTP1 administrator
	Recovers journal files	jnlmkrf	OpenTP1 administrator
	Integrates journals used for file restoration	jnlcolc	OpenTP1 administrator
	Copies an unloaded-journals file	jnlcopy	OpenTP1 administrator
	Displays archive status	jnlaris	General user
Displays edited unloaded-journals files or global archive unloaded-journals files	jnledit	OpenTP1 administrator	

	<b>Purpose</b>	<b>Command name</b>	<b>Access right</b>
	Outputs unloaded-journals file records or global archive unloaded-journals file records	jnlrput	OpenTP1 administrator
	Chronologically sorts and merges unloaded-journals files and global archive unloaded-journals files	jnlstts	OpenTP1 administrator
	Outputs statistics on about operations	jnlstts	OpenTP1 administrator
	Outputs statistics about MCF operations	jnlmest	OpenTP1 administrator
	Forcibly releases resource group connection	jnlardis	OpenTP1 administrator
DAM file management	Initializes a physical file	damload	OpenTP1 administrator
	Displays logical file status	damlst	General user
	Adds a logical file	damadd	OpenTP1 administrator
	Separates a logical file from online processing	damrm	OpenTP1 administrator
	Logically shuts down a logical file	damhold	OpenTP1 administrator
	Releases logical file shutdown	damrles	OpenTP1 administrator
	Deletes a physical file	damdel	OpenTP1 administrator
	Backs up a physical file	dambkup	OpenTP1 administrator
	Recovers a physical file	damrstr	OpenTP1 administrator
	Recovers a logical file	damfrc	OpenTP1 administrator
	Sets a threshold for the number of cache blocks	damchdef	OpenTP1 administrator
	Acquires the number of cache blocks	damchinf	General user

B. Library Functions and Commands

	<b>Purpose</b>	<b>Command name</b>	<b>Access right</b>
TAM file management	Initializes a TAM file	tamcre	OpenTP1 administrator
	Displays TAM table status	tamls	General user
	Adds a TAM table	tamadd	OpenTP1 administrator
	Separates a TAM table from online processing	tamrm	OpenTP1 administrator
	Logically shuts down a TAM table	tamhold	OpenTP1 administrator
	Releases TAM table logical shutdown	tamrles	OpenTP1 administrator
	Loads a TAM table	tamload	OpenTP1 administrator
	Unloads a TAM table	tamunload	OpenTP1 administrator
	Deletes a TAM file	tamdel	OpenTP1 administrator
	Backs up a TAM file	tambkup	OpenTP1 administrator
	Recovers a TAM file	tamrstr	OpenTP1 administrator
	Recovers a TAM file	tamfrc	OpenTP1 administrator
	Converts the TAM locked resource name	tamlckls	General user
	Displays the synonym information for a hash-format TAM file and TAM table	tamhsls	OpenTP1 administrator
MCF Message queue file management	Displays queue group status	quels	General user
	Allocates an MCF message queue physical file	queinit	OpenTP1 administrator
	Deletes an MCF message queue physical file	querm	OpenTP1 administrator

	<b>Purpose</b>	<b>Command name</b>	<b>Access right</b>
Resource manager management	Displays resource manager information	trnlstrm	General user
	Registers a resource manager	trnlkrm	OpenTP1 administrator
	Creates a transaction control object file	trnmkobj	OpenTP1 administrator
Trace management	Outputs UAP trace	uatdump	General user
	Merges RPC trace	rpcmrg	General user
	Outputs RPC trace	rpcdump	General user
	Outputs the shared memory dump	usmdump	OpenTP1 administrator
Remote API management	Sets up the execution environment of the remote API facility	rapsetup	OpenTP1 administrator
	Automatically creates definitions to be used for the remote API facility	rapdfgen	OpenTP1 administrator
	Displays the status of a RAP-processing listener or a RAP-processing server	rapls	OpenTP1 administrator
Management of performance verification traces	Edits and outputs prf trace information files	prfed	OpenTP1 administrator
	Gets prf trace information files	prfget	OpenTP1 administrator
Real-time statistics service management	Edits and outputs information in RTS log files	rtsedit	General user
	Outputs real-time statistics to the standard output	rtsls	General user
	Sets up the execution environment for the real-time statistics service	rtsssetup	OpenTP1 administrator
	Changes the settings for real-time statistics	rtssstats	OpenTP1 administrator
Connection management	Displays connection status	mcftlscn	General user
	Establishes a connection	mcftactcn	OpenTP1 administrator

B. Library Functions and Commands

	<b>Purpose</b>	<b>Command name</b>	<b>Access right</b>
	Releases a connection	mcftdctcn	OpenTP1 administrator
	Switches connections	mcfotchcn	OpenTP1 administrator
	Displays the network status	mcftlsln	General user
	Starts accepting server-type connection establishment requests	mcftonln	OpenTP1 administrator
	Stops accepting server-type connection establishment requests	mcftofln	OpenTP1 administrator
	Displays the status of multi-processing of messages	mcftrlstrd	OpenTP1 administrator
Application management	Displays application status	mcfalsap	General user
	Shuts down an application program	mcfadctap	OpenTP1 administrator
	Releases application program shutdown	mcfactap	OpenTP1 administrator
	Resets the number for abnormal terminations	mcfaclcap	OpenTP1 administrator
	Displays application timer start requests	mcfalstap	General user
	Deletes a start request to the timer associated with an application program	mcfadltap	OpenTP1 administrator
Application program operation support	Starts an application program	mcfuevt	General user
Logical terminal management	Displays logical terminal status	mcftlsle	General user
	Shuts down a logical terminal	mcftdctle	OpenTP1 administrator
	Releases logical terminal shutdown	mcfactle	OpenTP1 administrator
	Skips the first item in a logical terminal message queue	mcfvspgle	OpenTP1 administrator
	Holds messages in a logical terminal output queue	mcfthldoq	OpenTP1 administrator



	<b>Purpose</b>	<b>Command name</b>	<b>Access right</b>
	Releases the hold on messages in an output queue	mcftrlsq	OpenTP1 administrator
	Deletes a logical terminal output queue	mcftdlqle	OpenTP1 administrator
	Starts obtaining a logical-terminal message journal	mcftactmj	OpenTP1 administrator
	Stops obtaining the logical-terminal message journal	mcftdctmj	OpenTP1 administrator
	Forcibly terminates continuous inquiry-response processing for a logical terminal	mcftendct	OpenTP1 administrator
	Starts a proxy terminal	mcftstalt	OpenTP1 administrator
	Stops a proxy terminal	mcftedalt	OpenTP1 administrator
Service group management	Displays service group status	mcftlssg	General user
	Shuts down a service group	mcftdctsg	OpenTP1 administrator
	Releases shutdown of a service group	mcftactsg	OpenTP1 administrator
	Holds messages in an input queue for a service group	mcfthldiq	OpenTP1 administrator
	Releases the hold on messages in an input queue	mcftrlsiq	OpenTP1 administrator
	Deletes an input queue of a service group	mcftdlqsg	OpenTP1 administrator
Service management	Displays service status	mcftlssv	General user
	Shuts down a service	mcftdctsv	OpenTP1 administrator
	Releases shutdown of a service	mcftactsv	OpenTP1 administrator
Session management	Starts a session	mcftactss	OpenTP1 administrator

	<b>Purpose</b>	<b>Command name</b>	<b>Access right</b>
	Exits a session	mcftdctss	OpenTP1 administrator
Buffer management	Displays buffer group use	mcftlsbuf	General user
Mapping management	Changes map file pathnames	dcmapchg	OpenTP1 administrator
	Displays loaded-resources of map file	dcmapls	OpenTP1 administrator
MCF queue management	Copies contents of input or output queues	mcftdmpqu	General user
MCF trace management	Forcibly swaps MCF trace files	mcftswptr	OpenTP1 administrator
	Starts acquiring MCF trace information	mcftstrtr	OpenTP1 administrator
	Stops acquiring MCF trace information	mcftstprr	OpenTP1 administrator
MCF statistics management	Edits MCF statistics	mcfreport	OpenTP1 administrator
	Outputs MCF statistics	mcfstats	OpenTP1 administrator
MCF communication service management	Partially stops the MCF communication service	mcftstop	OpenTP1 administrator
	Partially starts the MCF communication service	mcftstart	OpenTP1 administrator
	References the status of the MCF communication service	mcftlscom	OpenTP1 administrator
User timer management	Displays the user timer monitoring status	mcftlsutm	General user

#### Note

There are also commands specific to the particular protocol. For protocol-specific commands, see the applicable *OpenTP1 Protocol* manual.

#### #

When executing the `dcstop` command from a UAP, execute the command in the background.

---

## C. Version Changes

---

This appendix describes the changes between versions under the following categories:

- Additions and deletions to functions, definitions, and commands
- Operational changes
- Changes to defaults for functions, definitions, and commands

### C.1 Changes in 07-03

The following table lists the additions and deletions to functions, definitions, and commands in TP1/Server Base 07-03.

*Table C-1: Additions and deletions to functions, definitions, and commands in TP1/Server Base 07-03*

Change	Category	Item added or deleted
Addition	Function	None
	Definition	System common definition <ul style="list-style-type: none"> <li>• <code>fil_prf_trace_delay_time</code> operand</li> <li>• <code>fil_prf_trace_option</code> operand</li> <li>• <code>jnl_prf_event_trace_level</code> operand</li> <li>• <code>uap_trace_file_put</code> operand</li> </ul>
		Lock service definition <ul style="list-style-type: none"> <li>• <code>lck_prf_trace_level</code> operand</li> </ul>
		Name service definition <ul style="list-style-type: none"> <li>• <code>name_cache_validity_time</code> operand</li> </ul>
		RAP-processing listener service definition <ul style="list-style-type: none"> <li>• <code>ipc_socketl_highwater</code></li> <li>• <code>ipc_socketl_watchtime</code></li> </ul>
		JNL performance verification trace definition
		LCK performance verification trace definition

C. Version Changes

Change	Category	Item added or deleted	
		Real-time acquisition item definition <ul style="list-style-type: none"> <li>• rts_mcf_ap_scd_stay operand</li> <li>• rts_mcf_ap_usr_srvc operand</li> <li>• rts_mcf_in_msg_scd_wait operand</li> <li>• rts_mcf_out_msg_norm_scd_wait operand</li> <li>• rts_mcf_out_msg_prio_scd_wait operand</li> <li>• rts_mcf_out_msg_resp_scd_wait operand</li> <li>• rts_mcf_out_msg_sync_scd_wait operand</li> <li>• rts_mcf_que_scd_wait_num operand</li> </ul>	
		User service default definition <ul style="list-style-type: none"> <li>• uap_trace_file_put operand</li> </ul>	
		User service definition <ul style="list-style-type: none"> <li>• uap_trace_file_put operand</li> </ul>	
	Command	filstatfs command <ul style="list-style-type: none"> <li>• -S option</li> </ul>	
		prctctrl command	
		prfed command <ul style="list-style-type: none"> <li>• -v option</li> </ul>	
		prfget command <ul style="list-style-type: none"> <li>• _mc, _fl, _jl, and _lk have been added as -f option values.</li> </ul>	
		uatdump command <ul style="list-style-type: none"> <li>• -f option</li> </ul>	
	Deletion		None

The following table lists the additions and deletions to functions, definitions, and commands in TP1/Message Control 07-03 and TP1/NET/Library 07-04.

*Table C-2: Additions and deletions to functions, definitions, and commands in TP1/Message Control 07-03 and TP1/NET/Library 07-04*

Change	Category	Item added or deleted
Addition	Function	None
	Definition	None
	Command	mcfalstap command
mcftlsutm command		
Deletion		None

The following table lists the operational changes in TP1/Server Base 07-03.

*Table C-3: Operational changes in TP1/Server Base 07-03*

Category	Operational change
Definition	System common definition <ul style="list-style-type: none"> <li>• The service information prioritizing function can now be specified in the <code>all_node</code> operand.</li> <li>• A priority selection node definition file has been added to the domain definition file whose usage is determined by the <code>name_domain_file_use</code> operand.</li> </ul>
	The number of performance verification trace acquisition services that can run and the related definitions have been changed.
Command	The <code>prctee</code> process that redirects OpenTP1's standard output and standard error output can now be terminated and restarted.
	The output format of the following command has been changed: <ul style="list-style-type: none"> <li>• <code>namsvinf</code></li> </ul>
	Areas in use and unused areas (available areas) can now be listed as user area information for the OpenTP1 file system.
	The results of editing trace information files can now be output as CSV files.
Message	KFCA26954-W, KFCA26956-W, KFCA26965-E, and KFCA27790-W <ul style="list-style-type: none"> <li>• The sender's IP address and port number have been added to the output information.</li> </ul>
Other	The following event trace information is now collected: <ul style="list-style-type: none"> <li>• FIL event trace</li> <li>• JNL performance verification trace</li> <li>• LCK performance verification trace</li> </ul>
	A UAP trace can now be collected without having to abort a process.
	The formula for estimating the static common memory has been changed.
	The formula for estimating the size of the shared memory pool has been changed.
	The following monitored events have been added: <ul style="list-style-type: none"> <li>• Start of OpenTP1 service</li> <li>• Stop of OpenTP1 service</li> </ul>
	In the formula for estimating the resources required for the UNIX message transmission functions, the formula for estimating the archive source node for the following resources has been changed: <ul style="list-style-type: none"> <li>• Message ID</li> <li>• Maximum number of queued messages in all OpenTP1 messages</li> </ul>

The following table lists the operational changes in TP1/Message Control 07-03 and

TP1/NET/Library 07-04.

*Table C-4: Operational changes in TP1/Message Control 07-03 and TP1/NET/Library 07-04*

Category	Operational change
Definition	The default value that is assumed when 0 or nothing is specified in the <code>-m</code> option of the <code>mcfatalcle</code> definition command has been changed from unlimited to 65535.
	The maximum value of the <code>-j</code> option has been changed from 131072 to 4000000 in the following definition commands: <ul style="list-style-type: none"> <li>• <code>mcfmcomn</code></li> <li>• <code>mcfmuap</code></li> <li>• <code>mcfcomn</code></li> </ul>
Other	The maximum number of MCF dump files that can be output has been changed from 99 to 3.
	The status of application timer start requests can now be displayed.
	The status of user timer monitoring can now be displayed.

The following table lists the changes to defaults in TP1/Server Base 07-03.

*Table C-5: Changes to defaults in TP1/Server Base 07-03*

Category	Changed default
Definition	Transaction service definition <ul style="list-style-type: none"> <li>• For the AIX version of uCosminexus TP1/Server Base(64), the default value of the <code>thread_stack_size</code> operand has been changed to 65536.</li> </ul>
	TAM service definition <ul style="list-style-type: none"> <li>• The default value of the <code>tam_pool_attri</code> operand has been changed as follows: <ul style="list-style-type: none"> <li>• In HP-UX or Solaris: <code>fixed</code></li> <li>• In AIX, Linux, or Windows: <code>free</code></li> </ul> </li> </ul>

There are no changes to defaults for functions, definitions, and commands in TP1/Message Control 07-03 or TP1/NET/Library 07-04.

## C.2 Changes in 07-02

The following table lists the additions and deletions to functions, definitions, and commands in TP1/Server Base 07-02.

*Table C-6: Additions and deletions to functions, definitions, and commands in TP1/Server Base 07-02*

Change	Category	Item added or deleted
Addition	Function	<code>dc_log_audit_print</code>

Change	Category	Item added or deleted
		CBLDCADT( 'PRINT' )
	Definition	<p>System common definition</p> <ul style="list-style-type: none"> <li>• nam_prf_trace_level operand</li> <li>• jnl_fileless_option operand</li> </ul> <p>XA resource service definition</p> <ul style="list-style-type: none"> <li>• xar_prf_trace_level operand</li> </ul> <p>System journal service definition</p> <ul style="list-style-type: none"> <li>• jnl_max_file_dispersion operand</li> <li>• jnl_min_file_dispersion operand</li> <li>• -e option of the jnladdpf definition command</li> </ul> <p>Log service definition</p> <ul style="list-style-type: none"> <li>• log_audit_out operand</li> <li>• log_audit_path operand</li> <li>• log_audit_size operand</li> <li>• log_audit_count operand</li> <li>• log_audit_message operand</li> </ul> <p>RAP-processing listener service definition</p> <ul style="list-style-type: none"> <li>• rap_term_disconnect_time operand</li> <li>• rap_stay_watch_time operand</li> <li>• rap_stay_warning_interval operand</li> <li>• log_audit_out_suppress operand</li> <li>• log_audit_message operand</li> <li>• watch_time operand</li> </ul> <p>RAP-processing client manager service definition</p> <ul style="list-style-type: none"> <li>• log_audit_out_suppress operand</li> <li>• log_audit_message operand</li> </ul> <p>Performance verification trace definition</p> <ul style="list-style-type: none"> <li>• prf_trace_backup operand</li> </ul> <p>XAR performance verification trace definition</p> <p>Real-time statistics service definition</p> <ul style="list-style-type: none"> <li>• rts_log_file_backup operand</li> </ul> <p>Definition of items logged in real-time</p> <ul style="list-style-type: none"> <li>• rts_scd_svc_scd_wait operand</li> <li>• rts_scd_svc_using_buf operand</li> <li>• rts_scd_parallel operand</li> </ul>

Change	Category	Item added or deleted		
		User service default definition <ul style="list-style-type: none"> <li>• log_audit_out_suppress operand</li> <li>• log_audit_message operand</li> <li>• scdsvcdef definition command</li> </ul>		
		User service definition <ul style="list-style-type: none"> <li>• log_audit_out_suppress operand</li> <li>• log_audit_message operand</li> <li>• <i>UAP-shared-library-name</i> was added to the value specified in the service operand</li> <li>• scdsvcdef definition command</li> </ul>		
	Command	dcauditsetup command		
		dcsetup command <ul style="list-style-type: none"> <li>• -j option</li> </ul>		
		prfget command <ul style="list-style-type: none"> <li>• _nm, _xr, and _pr were added to the values that can be specified in the -f option</li> </ul>		
		scdls command <ul style="list-style-type: none"> <li>• -ae option</li> <li>• -e option</li> <li>• -t option</li> </ul>		
		Deletion	Definition	User service default definition <ul style="list-style-type: none"> <li>• thdlock_sleep_time operand</li> </ul>
				User service definition <ul style="list-style-type: none"> <li>• thdlock_sleep_time operand</li> </ul>

The following table lists the additions and deletions to functions, definitions, and commands in TP1/Message Control 07-02 and TP1/NET/Library 07-03.

*Table C-7:* Additions and deletions to functions, definitions, and commands in TP1/Message Control 07-02 and TP1/NET/Library 07-03

Change	Category	Item added or deleted
Addition	Function	dc_mcf_adltap
		dc_mcf_tactcn
		dc_mcf_tactle
		dc_mcf_tdctcn
		dc_mcf_tdctle



Change	Category	Item added or deleted
		dc_mcf_tdlqle
		dc_mcf_tlscn
		dc_mcf_tlscom
		dc_mcf_tlsle
		dc_mcf_tslsln
		dc_mcf_tofln
		dc_mcf_tonln
		CBLDCMCF('ADLTAP')
		CBLDCMCF('TACTCN')
		CBLDCMCF('TACTLE')
		CBLDCMCF('TDCTCN')
		CBLDCMCF('TDCTLE')
		CBLDCMCF('TDLQLE')
		CBLDCMCF('TLSCOM')
		CBLDCMCF('TLSCN')
		CBLDCMCF('TLSLE')
		CBLDCMCF('TSLN')
		CBLDCMCF('TOFLN')
		CBLDCMCF('TONLN')
	Definition	MCF manager common definition <ul style="list-style-type: none"> <li>-i option in the mcfmcomn definition command</li> </ul>
	Command	mcfsslsg command <ul style="list-style-type: none"> <li>-m option</li> </ul>
Deletion		None

The following table lists the operational changes in TP1/Server Base 07-02.

Table C-8: Operational changes in TP1/Server Base 07-02

Category	Operational change
Definition	System common definition and system service common information definition <ul style="list-style-type: none"> <li>The minimum value of the <code>thdlock_sleep_time</code> operand was changed from 15 to 1.</li> </ul>
	System common definition and system service common information definition <ul style="list-style-type: none"> <li>The minimum value of the <code>thdlock_sleep_time</code> operand was changed from 15 to 1.</li> </ul>
	Process service definition <ul style="list-style-type: none"> <li>Additional events are acquired when <code>Y</code> (default) is specified in the <code>prc_prf_trace</code> operand.</li> </ul>
	Transaction service definition <ul style="list-style-type: none"> <li>When using the MCF service, the range of values that can be specified in the <code>trn_tran_process_count</code> operand (1-8192) becomes 1-7484 for the 32-bit version, or 1-6893 for the 64-bit version.</li> </ul>
Command	<b>dcdefchk command</b> <ul style="list-style-type: none"> <li>The logic check (LOG-0011) was changed for the <code>log_syslog_elist</code> operand and <code>log_syslog_elist_rint</code> operand.</li> <li>A logic check (LOG-0011) was added to <code>DCSYSLOGCTYPE</code>.</li> <li>Logic checks (LOG-0013 and LOG-0014) were added to operands related to audit logs.</li> </ul>
	<b>dcrasget command</b> <ul style="list-style-type: none"> <li>In Linux, the file extension of files acquired with the <code>-c</code> option specified was changed from <code>.z</code> to <code>.gz</code>.</li> </ul>
	Message KFCA01141-E is output when any of the following commands is executed in non-journal operation mode: <ul style="list-style-type: none"> <li><code>jnl1s</code></li> <li><code>jnlunlfg</code></li> <li><code>jnlchgfg</code></li> <li><code>jnlpnfg</code></li> <li><code>jnlclsfg</code></li> <li><code>jnlswpfg</code></li> <li><code>jnlrinf</code></li> <li><code>jnlarls</code></li> <li><code>jnlunl</code></li> <li><code>jnladdpf</code></li> <li><code>jnldepf</code></li> </ul>
	<b>usmdump command</b> <ul style="list-style-type: none"> <li>In Linux, the file extension of the shared memory dump file was changed from <code>.z</code> to <code>.gz</code>.</li> </ul>
Other	Message KFCA02512-E is output if the size of the backup file specified in the <code>damrstr</code> command is found to be invalid.

Category	Operational change
	Message KFCA26209-E is output if the size of the restore source file specified in the <code>tamrstr</code> command is found to be invalid.
	In Linux (IPF), the system retries if message output to syslog fails.
	In Linux, the file extension of the shared memory dump file was changed from <code>.z</code> to <code>.gz</code> .
	The following event trace information can be obtained: <ul style="list-style-type: none"> <li>• XAR performance verification trace</li> <li>• NAM event trace</li> <li>• Process service event trace</li> </ul>
	Message KFCA27764-W is output when the number of requests from RAP-processing clients that are waiting for allocation to a RAP-processing server exceeds the value specified in the <code>rap_stay_warning_interval</code> operand in the RAP-processing listener service definition.
	A backup of the RTS log file is created when the real-time statistics service starts.

The following table lists the operational changes in TP1/Message Control 07-02 and TP1/NET/Library 07-03.

*Table C-9: Operational changes in TP1/Message Control 07-02 and TP1/NET/Library 07-03*

Category	Operational change
Other	If the input queue still contains messages after a specific period of time has elapsed during OpenTP1 termination processing, the KFCA16532-I message is now displayed for each service group and then the KFCA16533-I message is displayed when processing of the remaining messages is completed.
	If the input queue still contains messages after a specific period of time has elapsed during OpenTP1 termination processing, the KFCA16534-I message is now displayed for each logical terminal and then the KFCA16535-I message is displayed when processing of the remaining messages is completed.
	If protocol control is still not ready to be terminated after a specific period of time has elapsed during OpenTP1 termination processing, the KFCA16536-I message is now displayed for each MCF communication service and then the KFCA16537-I message is displayed when protocol control is ready to be terminated.
	MHP dynamic loading of service functions is now supported.
	When a queue is monitored or when the system is waiting for a response message from the remote system, the processing status can now be obtained from the log messages issued during monitoring processing and upon completion of the processing.

Category	Operational change
	The following operations can now be achieved by using library functions: <ul style="list-style-type: none"> <li>• Displaying the connection status and establishing and releasing connection</li> <li>• Displaying the status of server-type connection establishment requests, and starting and stopping request acceptance</li> <li>• Deleting application timer start requests</li> <li>• Displaying the status of a logical terminal, shutting down a logical terminal, releasing a logical terminal from shutdown status, and deleting an output queue</li> <li>• Acquiring the status of the MCF communication service</li> </ul>
	The maximum number of unprocessed receive messages can now be displayed.

The following table lists the changes to defaults in TP1/Server Base 07-02.

*Table C-10: Changes to defaults in TP1/Server Base 07-02*

Category	Changed default
Definition	System common definition <ul style="list-style-type: none"> <li>• The default value of the <code>client_uid_check</code> operand has been changed from <code>y</code> to <code>N</code> in AIX, Linux, and Solaris (the default value is <code>y</code> in HP-UX and Windows).</li> </ul>

There are no changes to defaults for functions, definitions, and commands in TP1/Message Control 07-02 or TP1/NET/Library 07-03.

### C.3 Changes in 07-01

The following table lists the additions and deletions to functions, definitions, and commands in TP1/Server Base 07-01.

*Table C-11: Additions and deletions to functions, definitions, and commands in TP1/Server Base 07-01*

Change	Category	Item added or deleted
Addition	Function	None
	Definition	Name service definition <ul style="list-style-type: none"> <li>• <code>name_nodeid_check_message</code> operand</li> </ul>
		XA resource service definition <ul style="list-style-type: none"> <li>• <code>xar_msdtc_use</code> operand</li> </ul>
	Command	<code>dcpplist</code> command
<code>dcdefchk</code> command <ul style="list-style-type: none"> <li>• <code>-l</code> option</li> <li>• <code>-c</code> option</li> <li>• <code>-w</code> option</li> <li>• <code>-e</code> option</li> </ul>		

Change	Category	Item added or deleted
		xarinit command <ul style="list-style-type: none"> <li>-s option</li> </ul>
		xarls command <ul style="list-style-type: none"> <li>-r option</li> </ul>
Deletion		None

The following table lists the additions and deletions to functions, definitions, and commands in TP1/Message Control 07-01 and TP1/NET/Library 07-01.

*Table C-12: Additions and deletions to functions, definitions, and commands in TP1/Message Control 07-01 and TP1/NET/Library 07-01*

Change	Category	Item added or deleted
Addition	Function	None
	Definition	User service definition <ul style="list-style-type: none"> <li>mcf_prf_trace operand</li> </ul>
		MCF manager definition <ul style="list-style-type: none"> <li>mcfmsvg definition command</li> </ul>
		MCF application definition <ul style="list-style-type: none"> <li>-N option of the modelname operand in the mcfaalcap definition command</li> </ul>
		MCF performance verification trace definition <ul style="list-style-type: none"> <li>prf_file_size operand</li> <li>prf_file_count operand</li> </ul>
		Definition of system service information <ul style="list-style-type: none"> <li>mcf_prf_trace operand</li> </ul>
		System service common information definition <ul style="list-style-type: none"> <li>mcf_prf_trace_level operand</li> </ul>
	Command	mcf1sln command
		mcf1ofln command
		mcf1onln command
Deletion		None

The following table lists the operational changes in TP1/Server Base 07-01.

Table C-13: Operational changes in TP1/Server Base 07-01

Category	Operational change
Definition	Definition of items logged in real-time <ul style="list-style-type: none"> <li>• The information acquired by the <code>rts_jnl_read</code> operand (number of times) was changed to number of times, maximum time, minimum time, and average time.</li> <li>• The information acquired by the <code>rts_jnl_write</code> operand (number of times) was changed to number of times, maximum time, minimum time, and average time.</li> </ul>
Command	<code>xarevtr</code> command <ul style="list-style-type: none"> <li>• The application server name is displayed for a connection from .NET Framework.</li> <li>• The displayed application server XID information (the first 28 bytes or less of <code>GTRID</code>) was increased to a maximum of 64 bytes.</li> <li>• The displayed application server XID information (the first 28 bytes or less of <code>BQUAL</code>) was increased to a maximum of 64 bytes.</li> </ul>
	<code>xarforce</code> command <ul style="list-style-type: none"> <li>• The maximum number of characters that can be specified as the OpenTP1 transaction ID in the <code>-t</code> option was increased from 56 to 80.</li> <li>• The maximum number of characters that can be specified as the client transaction ID in the <code>-u</code> option was increased from 256 to 280.</li> </ul>
	<code>xarls</code> command <ul style="list-style-type: none"> <li>• The maximum number of characters that can be displayed as the OpenTP1 transaction ID was increased from 56 to 80.</li> <li>• The maximum number of characters that can be displayed as the client transaction ID was increased from 256 to 280.</li> </ul>
	<code>rtscedit</code> command <ul style="list-style-type: none"> <li>• The version of the RTS log file was added to the information output by the <code>-m</code> option.</li> </ul>
	<code>xarfills</code> command <ul style="list-style-type: none"> <li>• The size of the recovery information that can be stored was added to the output result.</li> </ul>
	<code>xarls</code> command <ul style="list-style-type: none"> <li>• DID information, node ID information, and recovery information used by MSDTC linkage was added to the output result of the <code>-c</code> option.</li> </ul>
Other	The following commands are no longer valid when specifying a RAP-processing listener or the RTSSPP (extended real-time statistics service): <code>sdcchprc</code> , <code>sdcldhold</code> , <code>sdcdrles</code> , and <code>sdcdrsprc</code>
	Data compression was added to remote procedure calls using the XA resource service.

The following table lists the operational changes in TP1/Message Control 07-01 and TP1/NET/Library 07-01.

*Table C-14: Operational changes in TP1/Message Control 07-01 and TP1/NET/Library 07-01*

Category	Operational change
Other	An MCF performance verification trace information file is now collected in the following cases: <ul style="list-style-type: none"> <li>• Y is specified in the <code>prf_trace</code> operand in the system common definition, or this specification is omitted</li> <li>• 00000001 is specified in the <code>mcf_prf_trace_level</code> operand in the system service common information definition</li> </ul>
	MCF communication process identifiers can now be displayed.
	Server-type connection establishment requests can now be started or stopped manually.

There are no changes to defaults for functions, definitions, and commands in TP1/Server Base 07-01, TP1/Message Control 07-01, or TP1/NET/Library 07-01.

## C.4 Changes in 07-00

The following table lists the additions and deletions to functions, definitions, and commands in TP1/Server Base 07-00.

*Table C-15: Additions and deletions to functions, definitions, and commands in TP1/Server Base 07-00*

Change	Category	Item added or deleted
Addition	Function	<code>dc_rts_utrace_put</code>
		<code>CBLDCRTS( 'RTSPUT ' )</code>
	Definition	System environment definition <ul style="list-style-type: none"> <li>• <code>user_command_online_tplmng_r_id</code> operand<sup>#1</sup></li> </ul>
		System common definition <ul style="list-style-type: none"> <li>• <code>dcstart_wakeup_retry_count</code> operand<sup>#2</sup></li> <li>• <code>ipc_listen_sockbufset</code> operand</li> </ul>
		User service default definition <ul style="list-style-type: none"> <li>• <code>ipc_listen_sockbufset</code> operand</li> </ul>
		User service definition <ul style="list-style-type: none"> <li>• <code>ipc_listen_sockbufset</code> operand</li> </ul>
		Name service definition <ul style="list-style-type: none"> <li>• 2 was added to the values that can be specified in the <code>name_audit_conf</code> operand</li> <li>• <code>name_audit_watch_time</code> operand</li> <li>• <code>name_rpc_control_list</code> operand</li> </ul>

C. Version Changes

Change	Category	Item added or deleted
		Transaction service definition <ul style="list-style-type: none"> <li>• trn_completion_limit_time operand</li> </ul>
		RAP-processing listener service definition <ul style="list-style-type: none"> <li>• rap_message_id_change_level operand</li> </ul>
		Real-time statistics service definition
		Definition of items logged in real-time
	Command	dcdefchk command
		dcsvgdef command <ul style="list-style-type: none"> <li>• -t option</li> </ul>
		namblad command
		ntbtail command <sup>#3</sup>
		rtsedit command
		rtsls command
		rtsetup command
		rtstats command
		tplconsole command <sup>#3</sup>
Deletion	None	

#1

Not supported in Linux or Windows.

#2

Not supported in Windows.

#3

Supported only in Windows 07-00-03 or later. For details about this command, see the precautions about using the Windows version of OpenTP1, provided in HTML format with the program product.

The following table lists the additions and deletions to functions, definitions, and commands in TP1/Message Control 07-00 and TP1/NET/Library 07-00.



*Table C-16: Additions and deletions to functions, definitions, and commands in TP1/Message Control 07-00 and TP1/NET/Library 07-00*

Change	Category	Item added or deleted
Addition	Function	None
	Definition	MCF manager definition <ul style="list-style-type: none"> <li>order operand of the <code>-c</code> option in the <code>mcfmuap</code> definition command</li> </ul>
		DCMCFCMDLOG environment variable
	Command	None
Deletion	Function	None
	Definition	MCF manager definition <ul style="list-style-type: none"> <li>delayed operand of the <code>-t</code> option in the <code>mcfmcomn</code> definition command</li> <li><code>mcfmrclnt</code> definition command</li> <li><code>mcfmrerun</code> definition command</li> </ul>
		Remote MCF manager definition <ul style="list-style-type: none"> <li><code>mcfrcmn</code> definition command</li> <li><code>mcfrrserv</code> definition command</li> </ul>
		Definition of system service information <ul style="list-style-type: none"> <li>critical operand</li> </ul>
	Command	None

The following table lists the operational changes in TP1/Server Base 07-00.

*Table C-17: Operational changes in TP1/Server Base 07-00*

Category	Operational change
Definition	System environment definition <ul style="list-style-type: none"> <li>The maximum value of the <code>server_count</code> operand was increased from 2048 to 4096.<sup>#1</sup></li> </ul>
Command	<code>jnlcolc</code> , <code>jnlcopy</code> , <code>jnledit</code> , <code>jnlrput</code> , and <code>jnlSORT</code> commands <ul style="list-style-type: none"> <li>The number of unload files that can be specified was increased from 64 to 256.</li> </ul>
	<code>jnlEDIT</code> command <ul style="list-style-type: none"> <li>The creation date/time of the MCF record and the date/time at which the transaction branch was started were added to the output result.</li> </ul>
	<code>prctee</code> command <ul style="list-style-type: none"> <li>The message output destination was changed from <code>/tmp</code> to <code>\$DCDIR/spool</code>.</li> <li>The output destination for error messages was changed from <code>/tmp/betran.log</code> to <code>\$DCDIR/spool/.prctee.log</code>.</li> </ul>

Category	Operational change
	xarevtr command <ul style="list-style-type: none"> <li>The application server name is displayed in the output result for a connection from Cosminexus V6.5 or later.</li> </ul>
Message	KFCA00107-E <ul style="list-style-type: none"> <li>The information indicated in the message details was changed from the module name to the OpenTP1 file system name.</li> </ul>
	KFCA00854-E <ul style="list-style-type: none"> <li>Information about the message size that could not be stored was added.</li> </ul>
Other	The number of characters that can be specified in a host name was increased from 63 to 255.
	The command priority in the sample scenario template for JP1/AJS2-SO was changed from None to 3.
	The access mode setting for UNIX domain communication files was changed from 755 to 777. <sup>#2</sup>
	Japanese messages in UTF-8 (LANG ja_JP.UTF-8) are supported. <sup>#2</sup>

#1

Not supported in Linux.

#2

Supported in Linux only.

The following table lists the operational changes in TP1/Message Control 07-00 and TP1/NET/Library 07-00.

*Table C-18: Operational changes in TP1/Message Control 07-00 and TP1/NET/Library 07-00*

Category	Operational change
Definition	MCF communication configuration definition <ul style="list-style-type: none"> <li>The syntax element of the -a option in the mcftenv definition command was changed from an identifier of 1-8 characters to 1-8 alphanumeric characters.</li> </ul>
Command	A facility for collecting command logs was added to the following commands: mcfaactap, mcfaclcap, mcfadctap, mcfadltap, mcfstats, mcftactcn, mcftactle, mcftactmj, mcftactsg, mcftactsv, mcftchcn, mcftdctcn, mcftdctle, mcftdctmj, mcftdctsg, mcftdctsv, mcftdlqle, mcftdlqsg, mcftdmpqu, mcfthldiq, mcfthldoq, mcftrlsiq, mcftrlsq, mcftspqle, mcftstart, mcftstop, and mcfuevt

There are no changes to defaults for functions, definitions, and commands in TP1/

Server Base 07-00.

The following table lists the changes to defaults in TP1/Message Control 07-00 and TP1/NET/Library 07-00.

*Table C-19:* Changes to defaults in TP1/Message Control 07-00 and TP1/NET/Library 07-00

Category	Changed default
Definition	MCF communication configuration definition <ul style="list-style-type: none"><li data-bbox="639 443 1461 495">• The default for the <code>disk</code> operand in the <code>mcftrc</code> definition command was changed from <code>no</code> to <code>yes</code>.</li></ul>

---

## D. Overview of Remote Procedure Call Processing

---

This appendix provides an overview of remote procedure calls for each function and each status. All the information provided in this appendix relates to queue-receiving remote procedure calls.

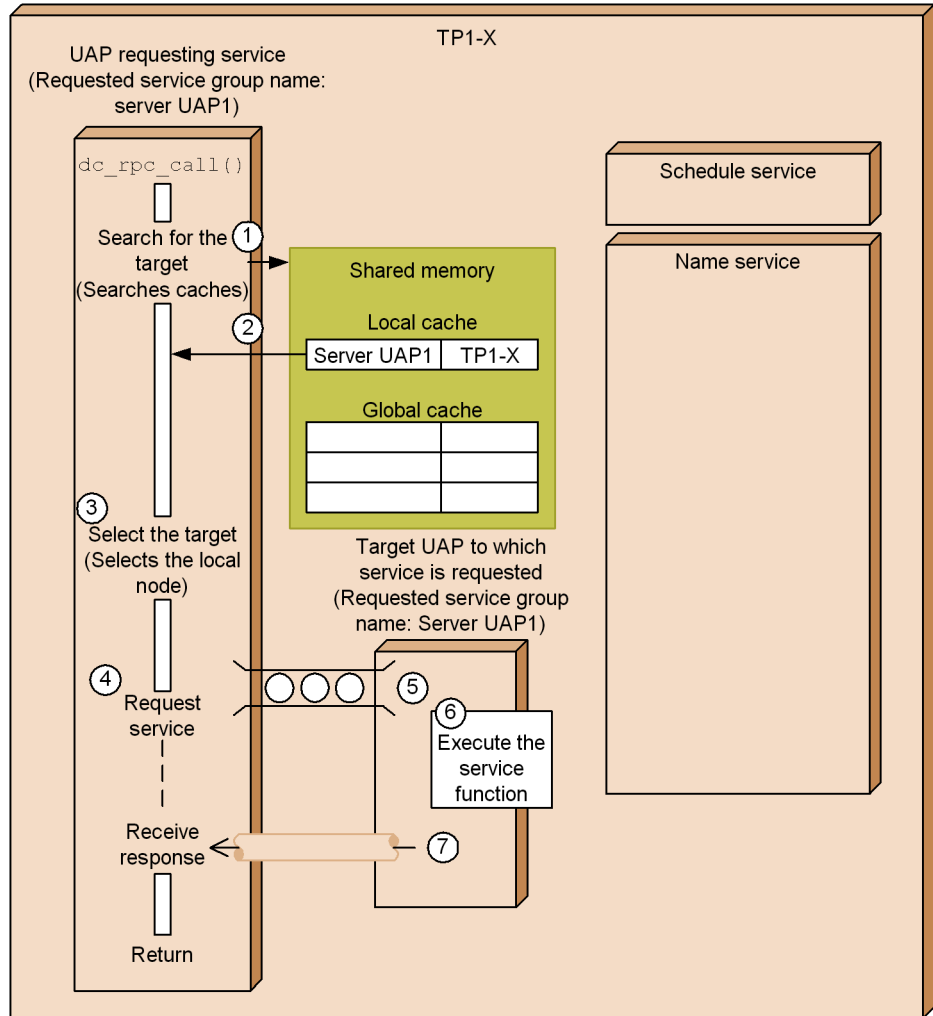
In this appendix, *local cache* refers to an area used to manage service information for the server that is run on the local OpenTP1 system by the name service; *global cache* refers to an area used to manage service information for the servers that are run at the remote nodes by the name service.

For details about how to create application programs, see the *OpenTP1 Programming Guide*. For details about the syntax for the functions, see the manual *OpenTP1 Programming Reference C Language* or *OpenTP1 Programming Reference COBOL Language*.

### D.1 Overview of processing a remote procedure call to the local node

The following figure provides an overview of processing a remote procedure call to the local node.

Figure D-1: Overview of processing a remote procedure call to the local node



Legend:

- : Flow of service information search
- : Inter-process communication processing
- ○ ○ : Message queue
- : Connection

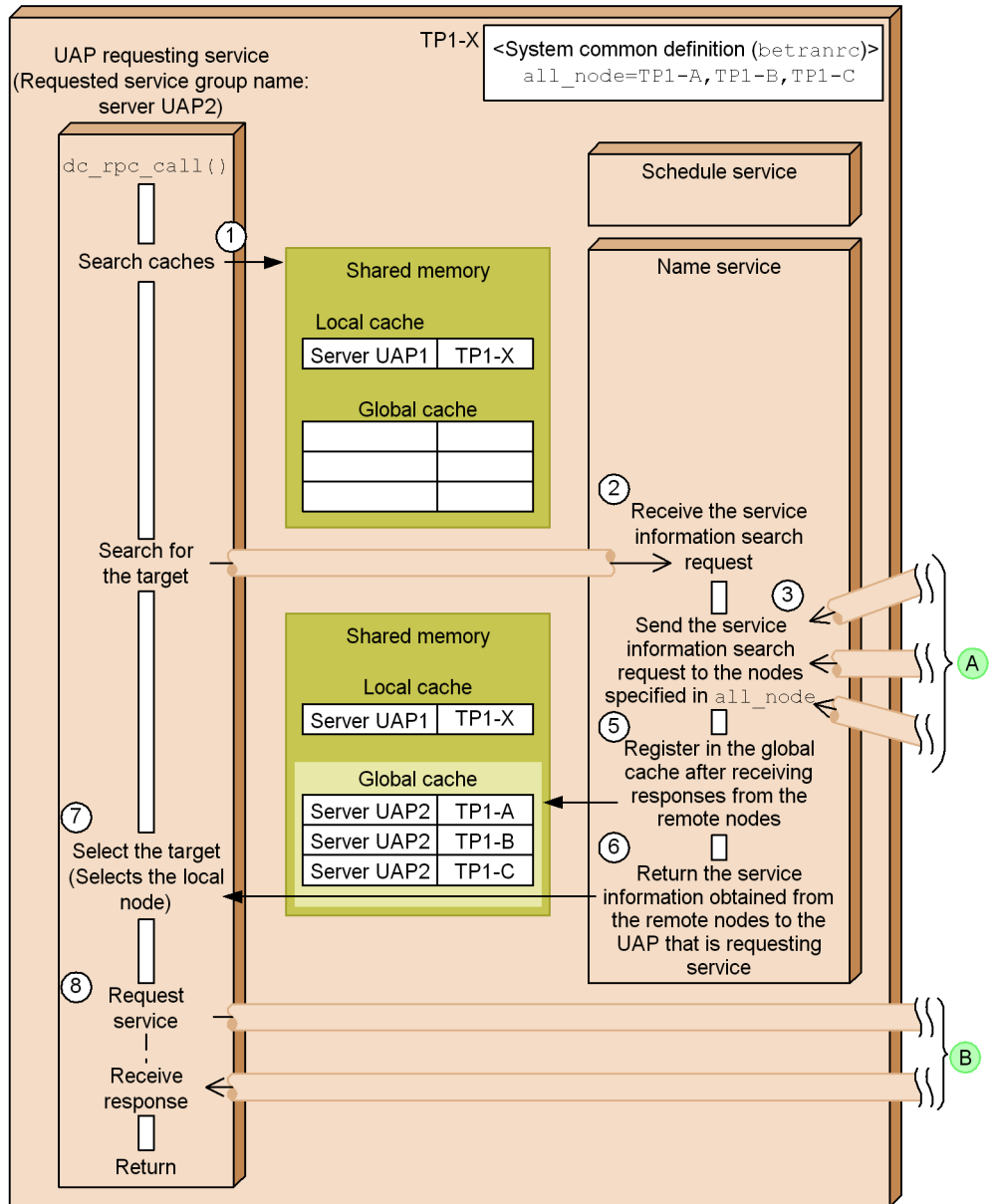
The flow of TP1-X processing shown in the figure is described below. The numbers correspond to the circled numbers in the figure.

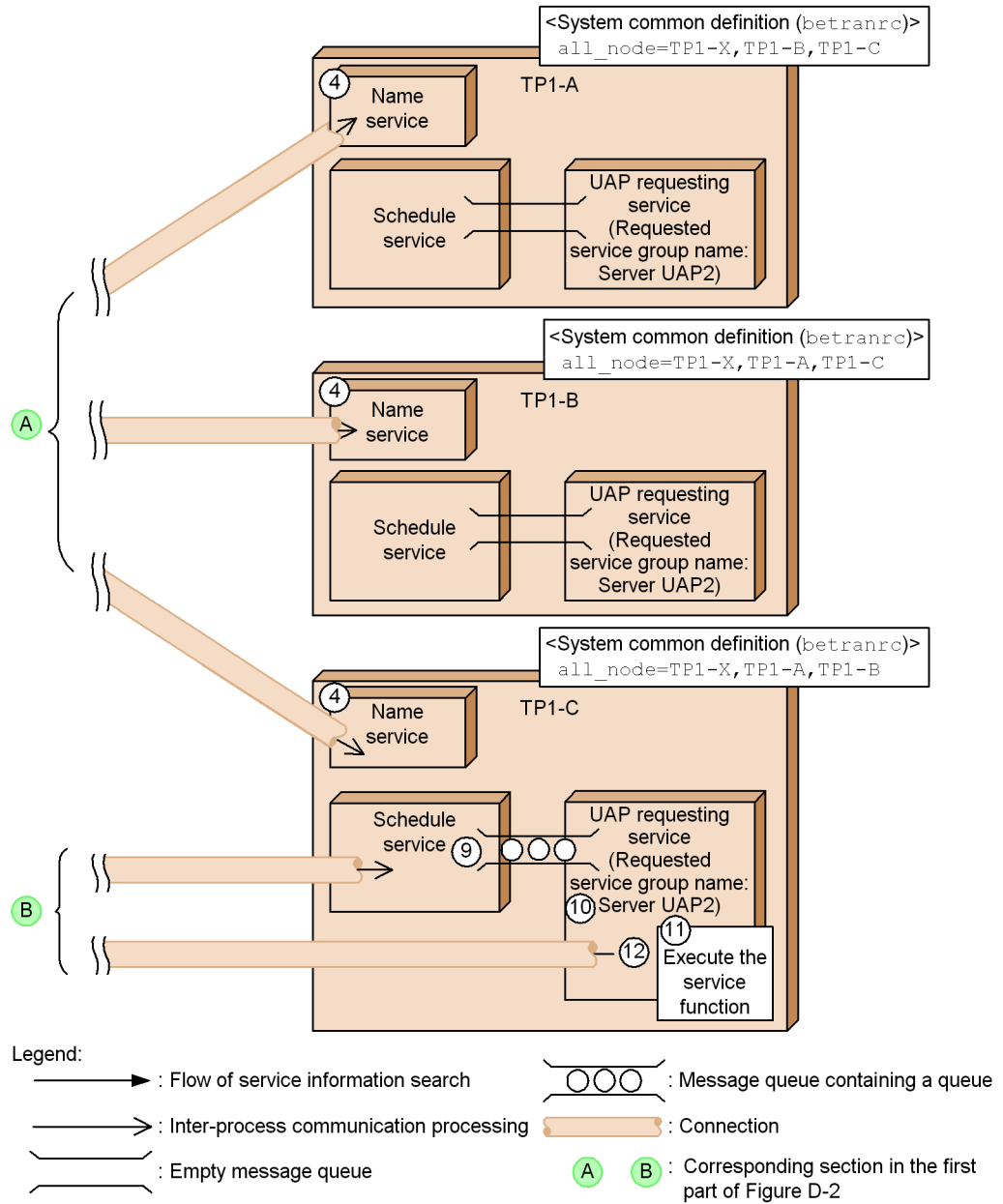
1. Checks whether the corresponding service information has been registered in the local and global caches, using as the search key the service group name that was specified in an argument of the `dc_rpc_call` function.
2. Because the corresponding service information was found only in the local cache, only the service information in the local cache is returned to the UAP requesting the service.
3. Because only the service information for the UAP running at the local node was obtained, TP1-X selects the UAP (server `UAP1`) running at the local node as the target to which the service is requested.
4. In the case of a service request to the UAP in the local OpenTP1 system, the service request is placed directly in the message queue without going through the schedule service.
5. The target UAP (server `UAP1`) for which the service request was registered retrieves the service request from the message queue.
6. Executes the service function.
7. After executing the service function, the target UAP (server `UAP1`) directly sends a response message to the UAP that issued the service request.

## **D.2 Overview of processing a remote procedure call to remote nodes**

The following figure provides an overview of processing a remote procedure call to remote nodes.

Figure D-2: Overview of processing a remote procedure call to remote nodes





The flow of processing shown in the figure is described below. The numbers correspond to the circled numbers in the figure. Numbers 1-3 and 5-8 describe TP1-X processing, number 4 describes TP1-A, TP1-B, and TP1-C processing, and numbers



9-12 describe TP1-C processing.

1. Checks whether the corresponding service information has been registered in the local and global caches by using as the search key the service group name that was specified in an argument of the `dc_rpc_call` function.
2. If there is no corresponding service information, a service information search request is sent to the name service.

If the local or global cache contains the corresponding service information, but the conditions listed below are met, the system deletes the service information from the global cache and then sends a service information search request to the name service:

- The global cache contains the corresponding service information, but that service information has expired.<sup>#1</sup>
  - This is the first service information search request issued from a UAP to the corresponding service after the UAP was started at the local node.
3. The name service of TP1-X that receives the service information search request sends out the request to the nodes (TP1-A, TP1-B, and TP1-C) specified in the `all_node` operand in the system common definition.  
 Note that if the `all_node` operand is omitted, the name service does not send the service information search request. If any of the target nodes specified in the `all_node` operand are registered in the RPC suppression list,<sup>#2</sup> the service information search request is not sent to those nodes.
  4. The name services of TP1-A, TP1-B, and TP1-C that receive the service information search request search their local caches for the corresponding service group and then send the results back to the name service of TP1-X.
  5. Upon receiving from the other nodes the responses to the service request, the name service of TP1-X registers the received service information in the global cache.
  6. The name service of TP1-X returns the service information obtained from the searches to the UAP that issued the service request.
  7. If multiple UAPs with the same service group name are running, one of the UAPs (server UAP2) is selected as the recipient of the service request by TP1's internal processing. In this example, TP1-C is selected as the recipient.
  8. A service execution request is sent to the schedule service of TP1-C that was selected in 7.
  9. The schedule service that receives the service request registers the service request in the message queue for the target UAP (server UAP2) in the corresponding service group.

10. The target UAP (server UAP2) for which the service request has been registered retrieves the service request from the message queue.
11. Executes the service function.
12. After the service function has executed, the UAP (server UAP2) that received the service request sends a response message directly to the UAP that issued the service request.

#1

This is the amount of elapsed time from when the service information was acquired to the time specified in the `name_cache_validity_time` operand in the name service definition.

For details about the `name_cache_validity_time` operand, see the manual *OpenTP1 System Definition*.

#2

This list contains information about inactive nodes.

If communication with another node's name server fails, that node is registered in the RPC suppression list. Once a node is registered in the RPC suppression list, all service information for that node is deleted from the global cache. If the node monitoring function detects that a node's status has changed from inactive to active, the information about that node is then deleted from the RPC suppression list.

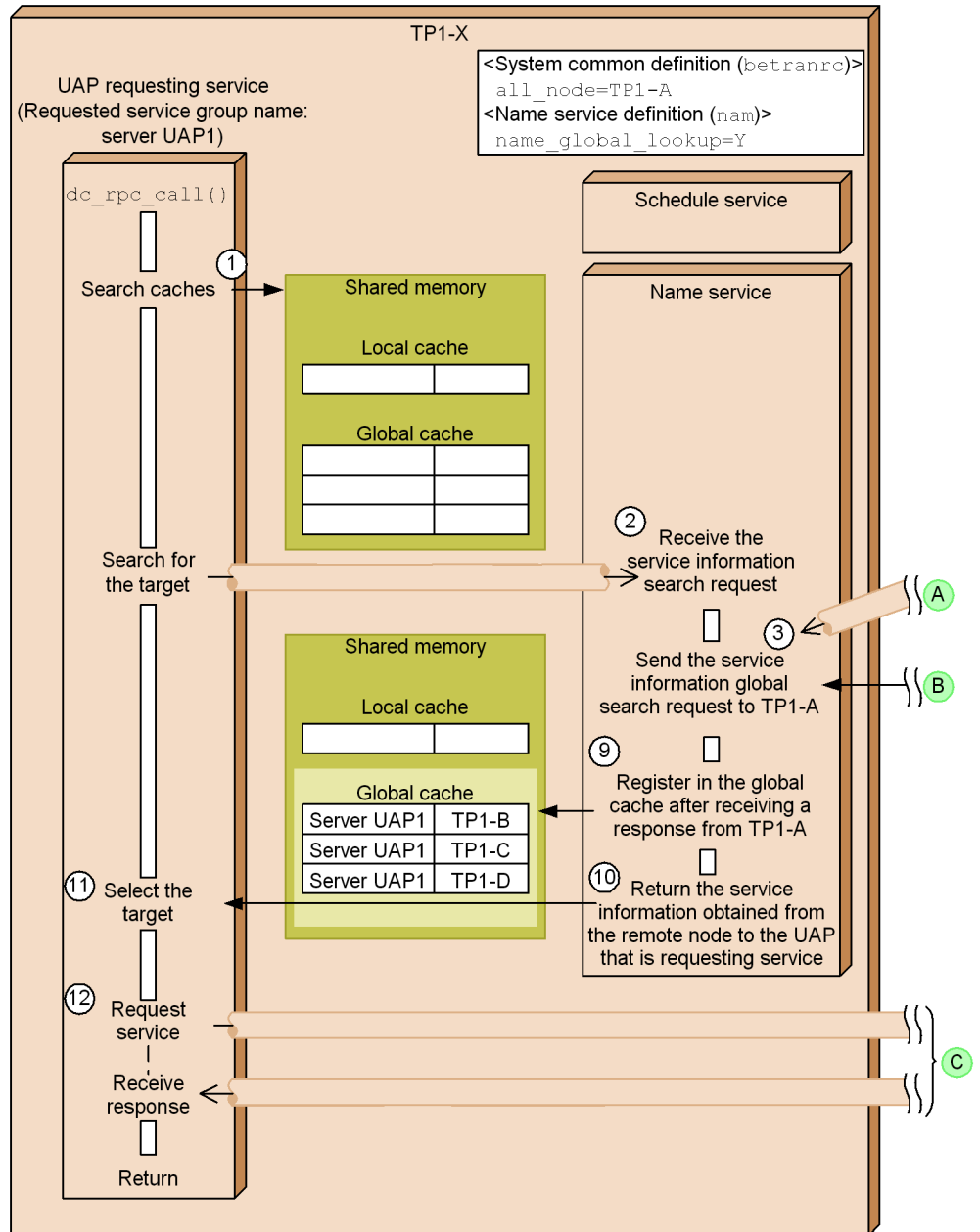
For details, see *3.2.3(3) Facility for monitoring nodes registered in the RPC suppression list*.

### D.3 Overview of global search processing

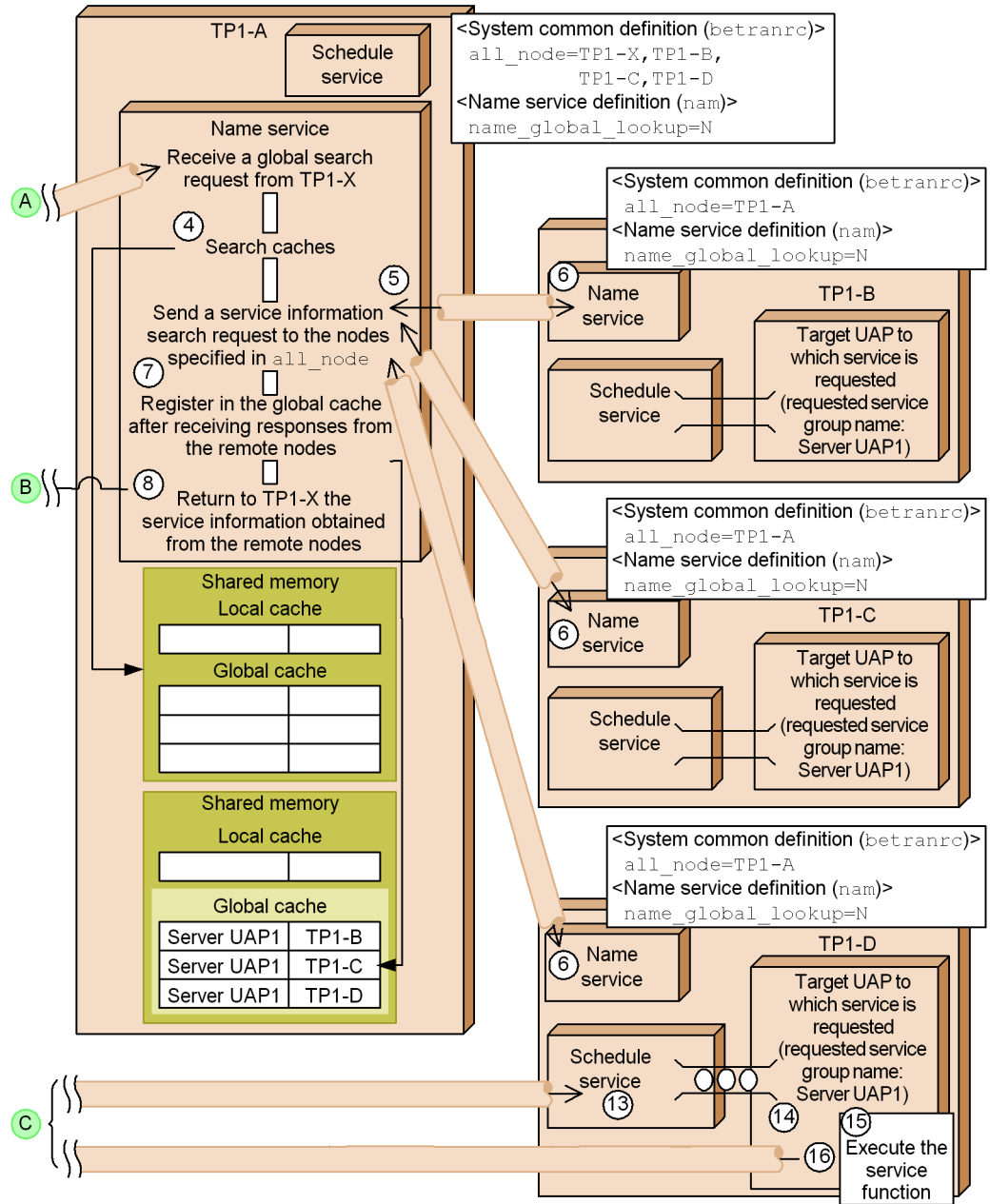
Global search is a service search method employed at the local OpenTP1 system to search the nodes specified in the `all_node` operand and the nodes specified in the `all_node` operands at the corresponding nodes. For details about the global search facility, see *3.2.2(1) Global search facility*.

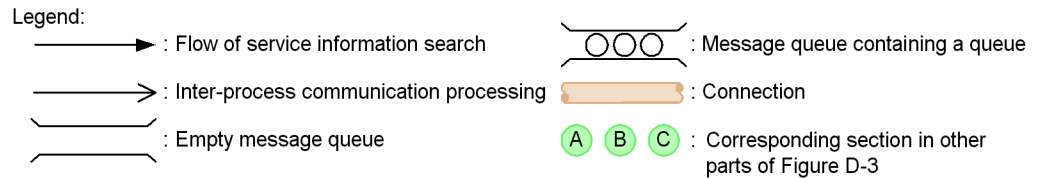
The figure below provides an overview of global search processing. For details about the service information search conditions, see *D.2 Overview of processing a remote procedure call to remote nodes*.

Figure D-3: Overview of global search processing



D. Overview of Remote Procedure Call Processing





The flow of processing shown in the figure is described below. The numbers correspond to the circled numbers in the figure. Numbers 1-3 and 9-12 describe TP1-X processing, numbers 4, 5, 7, and 8 describe TP1-A processing, number 6 describes TP1-B, TP1-C, and TP1-D processing, and numbers 13-16 describe TP1-D processing.

1. Checks whether the corresponding service information has been registered in the local and global caches by using as the search key the service group name that was specified in an argument of the `dc_rpc_call` function.
2. If there is no corresponding service information, a service information search request is sent to the name service.
3. TP1-X sends a service information global search request to the node (TP1-A) that was specified in the `all_node` operand in the system common definition.
4. The name service of TP1-A that receives the global search request searches its local and global caches for the corresponding service group.
5. If no applicable service information is found in the caches, TP1-A sends a service information search request to the nodes (TP1-B, TP1-C, and TP1-D) specified in the `all_node` operand in the system common definition.
6. The name services of TP1-B, TP1-C, and TP1-D that receive the service information search request search their local caches for the corresponding service group and then send the results back to the name service of TP1-A.
7. Upon receiving from the other nodes the responses to the service request, the name service of TP1-A registers the received service information in the global cache.
8. The name service of TP1-A sends the service information obtained from the searches to the name service of the TP1-X node that issued the service request.
9. Upon receiving the responses to the global service search request from TP1-A, the name service of TP1-X registers the received service information in the global cache.
10. The name service of TP1-X returns the service information obtained from the searches to the UAP that issued the service request.
11. If multiple UAPs with the same service group name are running, one of the UAPs

(server UAP1) is selected as the recipient of the service request. In this example, TP1-D is selected as the recipient.

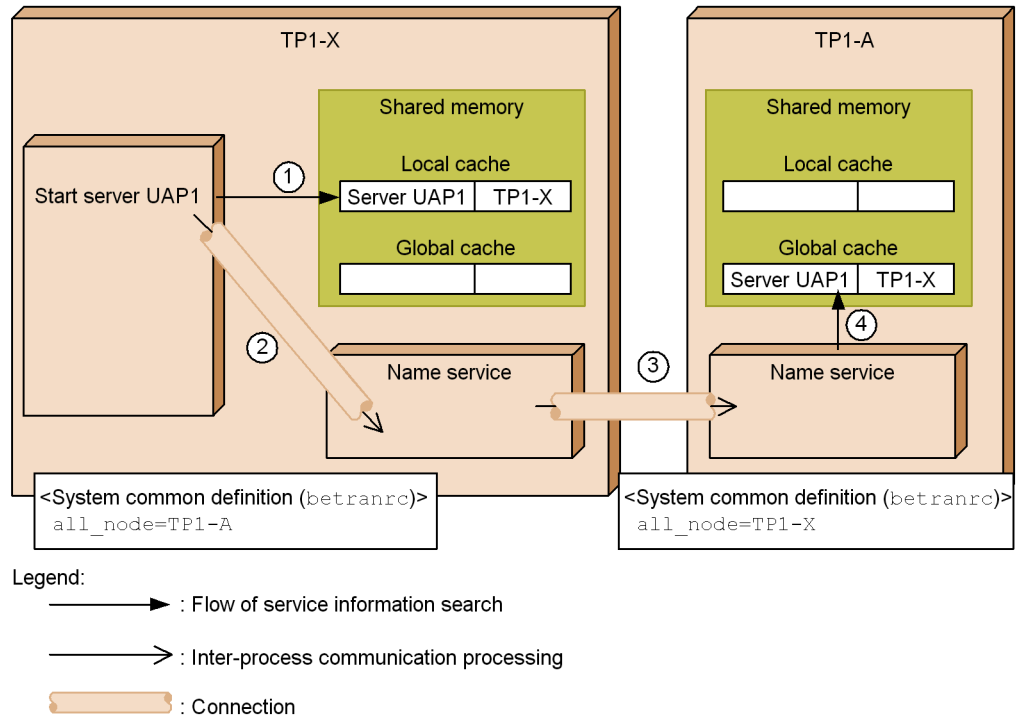
12. A service execution request is sent to the schedule service of TP1-D that was selected in 11.
13. The schedule service that receives the service request registers it in the message queue for the target UAP (server UAP1) in the corresponding service group.
14. The target UAP (server UAP1) for which the service request has been registered retrieves the service request from the message queue.
15. Executes the service function.
16. After the service function has executed, the UAP (server UAP1) that received the service request sends a response message directly to the UAP that issued the service request.

#### **D.4 Overview of service information registration and deletion processing**

This section provides an overview of the processing for service information registration and deletion.

The following figure provides an overview of service information registration processing.

Figure D-4: Overview of service information registration processing



The flow of processing shown in the figure is described below. The numbers correspond to the circled numbers in the figure. Numbers 1-3 describe TP1-X processing, and number 4 describes TP1-A processing.

1. When the server UAP1 starts, the service information for UAP1 is registered in the local cache in the local OpenTP1 system (TP1-X).
2. A service information registration request is sent to the name service of TP1-X.
3. Upon receiving the service information registration request, the name service sends the service information registration request to the name service of TP1-A that is specified in the `all_node` operand in the system common definition.
4. Upon receiving the service information registration request, the name service of TP1-A checks the `all_node` operand in the system common definition for the transmission source.

If the transmission source is specified in the `all_node` operand, the name service of TP1-A registers in the global cache the service information for the server UAP1 as requested.

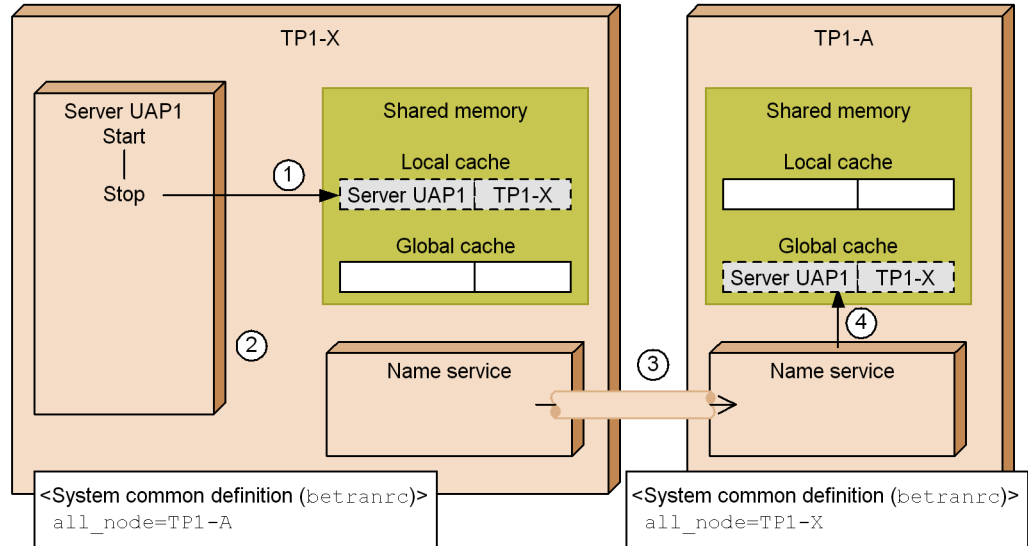
If the transmission source is not specified in the `all_node` operand, the name

service of TP1-A does not register the service information.

In this figure, service information for the server UAP1 is registered in the global cache because the transmission source TP1-X is specified in the `all_node` operand.

The following figure provides an overview of service information deletion processing.

Figure D-5: Overview of service information deletion processing



Legend:

- : Flow of service information search
- : Inter-process communication processing
- : Connection

The flow of processing shown in the figure is described below. The numbers correspond to the circled numbers in the figure. Numbers 1-3 describe TP1-X processing, and number 4 describes TP1-A processing.

1. When the server UAP1 is terminated, its service information is deleted from the local cache in the local OpenTP1 (TP1-X) system.
2. A service information deletion request is sent to the name service of TP1-X.
3. Upon receiving the service information deletion request, the name service sends it to the name service of TP1-A, which is specified in the `all_node` operand in the system common definition.
4. Upon receiving the service information deletion request, the name service of



TP1-A checks the `all_node` operand in the system common definition for the transmission source.

If the transmission source is specified in the `all_node` operand, the name service of TP1-A deletes from the global cache the service information for the server UAP as requested.

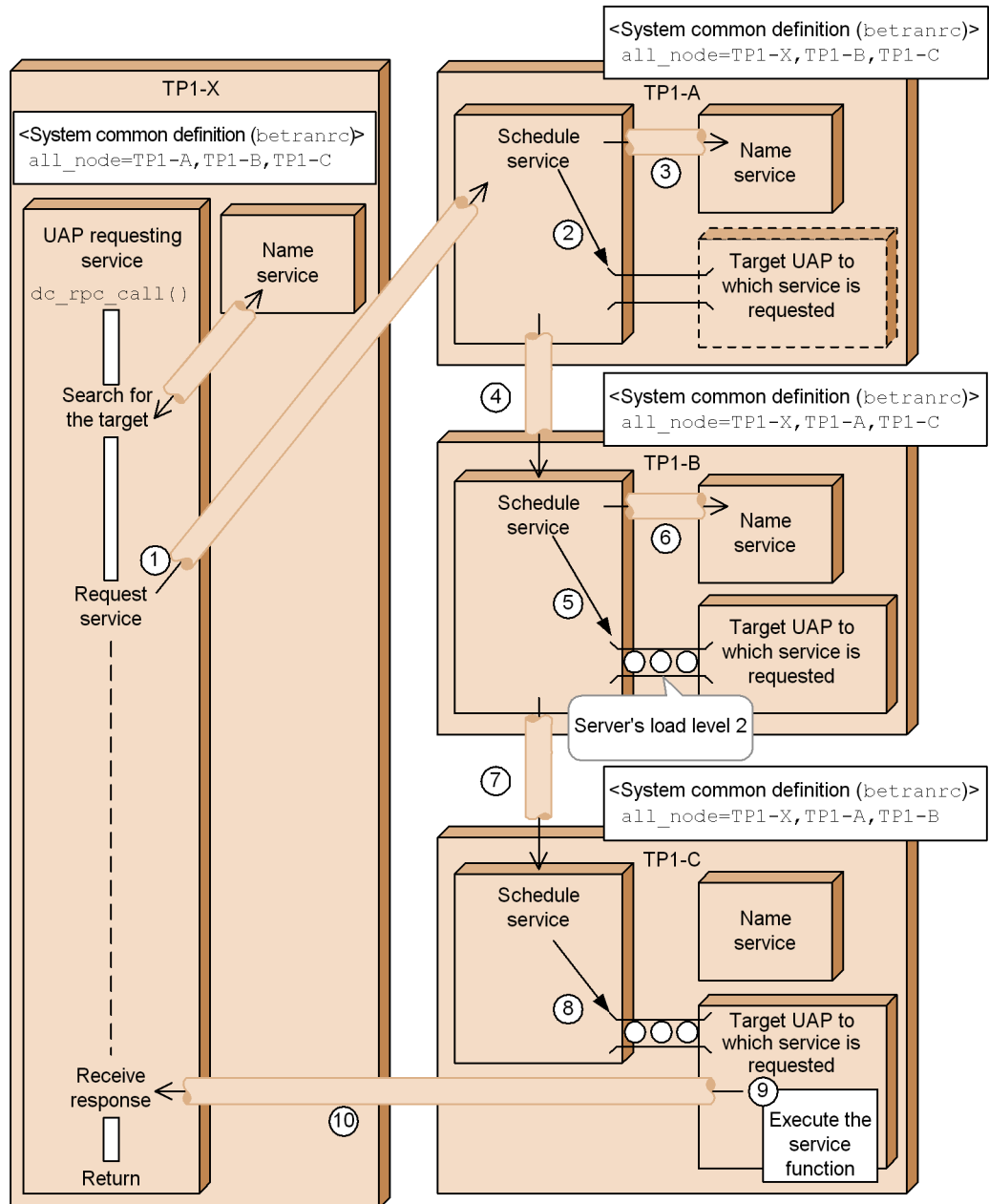
If the transmission source is not specified in the `all_node` operand, the name service of TP1-A does not delete the service information.

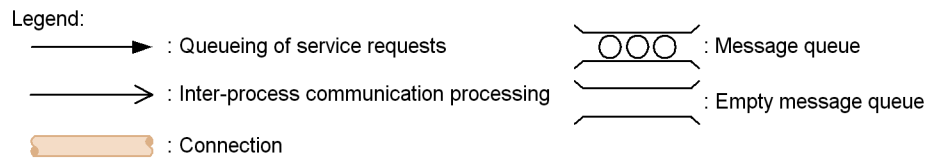
In this figure, service information for the server UAP1 is deleted from the global cache because the transmission source TP1-X is specified in the `all_node` operand.

## D.5 Overview of node-to-node forwarding processing

The following figure provides an overview of node-to-node forwarding processing.

Figure D-6: Overview of node-to-node forwarding processing





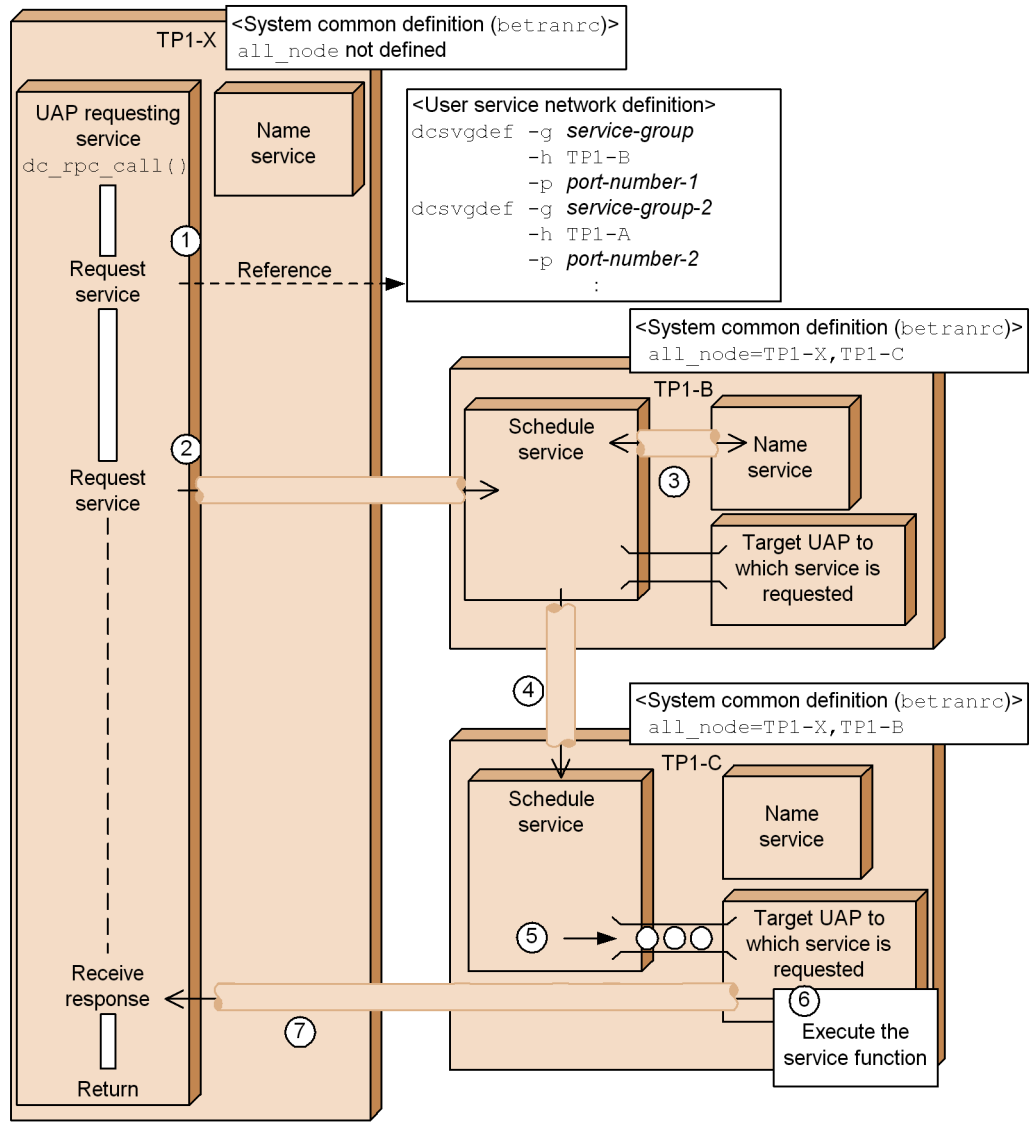
The flow of processing shown in the figure is described below. The numbers correspond to the circled numbers in the figure. Number 1 describes TP1-X processing, and numbers 2-10 describe TP1-A, TP1-B, and TP1-C processing.

1. As a result of a name service search, TP1-X sends a service request to the schedule service of node TP1-A.
2. This service request is not queued because the server of TP1-A was shut down when the schedule service of TP1-A attempted to issue the service request.
3. The schedule service searches the name service for a forwarding destination.  
If the RPC request of the program that issued the service request is using the scheduler direct facility with the `dc_rpc_call_to` function, the schedule service returns an error to that program without forwarding the request.
4. The service request is forwarded to the schedule service of TP1-B.
5. The service request is not queued because this server's load level is high (2).
6. The schedule service searches the name service for another forwarding destination.
7. The schedule service forwards the service request to the schedule service of TP1-C.
8. The service request is registered in the message queue because this server is not shut down nor is its load level high.
9. Executes the service function.
10. After the service function has executed, the UAP (server UAP1) that received the service request sends a response message directly to the UAP that issued the service request.

## D.6 Overview of remote procedure call processing using the `dcsvgdef` definition command

The following figure provides an overview of remote procedure call processing using the `dcsvgdef` definition command in the user service network definition.

Figure D-7: Overview of remote procedure call processing using the dcsvgdef definition command



The flow of processing shown in the figure is described below. The numbers correspond to the circled numbers in the figure. Numbers 1 and 2 describe TP1-X processing, numbers 3 and 4 describe TP1-B processing, and numbers 5-7 describe TP1-C processing.

1. TP1-X obtains the recipient of a service request (TP1-B) from the user service network definition without searching the name service.
2. The request is sent to the schedule service of TP1-B.
3. The schedule service running at TP1-B searches the name service and then returns a result indicating that the UAP exists at TP1-B and TP1-C.
4. The service request is forwarded to TP1-C, which is selected randomly.
5. The schedule service of TP1-C places the service request in the message queue.
6. Executes the service function.
7. After the service function has executed, the UAP (UAP1) that received the service request sends a response message directly to the UAP that issued the service request.

---

## E. Glossary

---

### **application**

A general name for a user-created program that is executed in an OpenTP1 system. An application that carries out user tasks is called a *user application program*. An application that registers a UAP in OpenTP1 and performs as a server is called a *user server*.

### **archived-journals file (or archive journal file)**

A file that contains the system journals from OpenTP1 nodes. In a cluster system or parallel-processing system, the Global Archive Journal facility is used to archive the system journals from multiple OpenTP1 nodes into a single archived-journals file. This simplifies management of the nodes because the OpenTP1 administrator needs to unload this single file only, rather than each of the system journal files on each node.

See also *global archive journal service*.

### **audit log**

A file containing historical information about the operations performed on OpenTP1 programs by system developers, operators, and users, together with the program behavior triggered by those operations. An audit log contains information such as what was done, when, and by whom. Thus, they can serve as reference material for auditing system usage, unauthorized access, and other such matters.

### **checkpoint dump**

Journals periodically collected into a file during OpenTP1 operations and used to reduce recovery time. Checkpoint dumps are taken at various checkpoints. At each checkpoint, OpenTP1 saves the status of various system tables that are needed in recovery. During recovery, the information in a checkpoint dump can replace all the recovery journals from the start of online processing to the most recent checkpoint. Using a checkpoint dump and only the most recent recovery journals reduces recovery time.

### **client gateway program (CGW)**

A program to enable OpenTP1 to access other open systems such as DCE. A CGW receives a request for a service from a UAP in OpenTP1, and it requests the same service from an open system. In an OpenTP1 system, a client gateway program is regarded as a service-providing program.

### **client UAP/server UAP**

A client UAP requests a service, and a server UAP carries out the requested service. A client UAP can be a service-using program, service-providing program, or message-handling program. A server UAP is a service-providing program.

**client user program (CUP)**

A client UAP that uses the OpenTP1 client facilities provided by the product TP1/Client. These facilities include useful troubleshooting facilities: the error log facility and the UAP trace facility.

**client/server**

A term describing a relationship between programs. A client is a program that requests a service; a server is a program that carries out the requested service. Whether a program is functioning as a client or a server is determined by its relationship to another program.

**cluster system or parallel-processing system**

A system in which multiple processors, perhaps on different machines, are so connected that they can work together to provide a single service. In such a system, one logical task might be broken down into different fragments and all the fragments might be processed simultaneously on different processors. Such system configurations can provide very high performance. The product TP1/Multi supports OpenTP1 operations in a cluster system or parallel-processing system.

**commit**

A commit operation ensures successful completion of a transaction at a synchronization point. When a commit operation is complete (i.e., when a transaction completes synchronization point processing), the results of the transaction take effect. See also *transaction* and *rollback*.

**complete recovery**

Returning an entire OpenTP1 system to its former state after a system failure.

**connection**

A logical communication path established between an OpenTP1 system and another system. Note that some communication protocols also use the term "association" with the same meaning. In this manual, a connection is named consistently as long as the message exchange is described.

**CUP**

See *client user program (CUP)*.

**DAM**

See *direct access method file (DAM file)*.

**deadlock**

Conflict situation in which more than one UAP tries to reserve more than one resource at the same time, and each UAP continues to wait for another to release one of the resources.

### **direct access method file (DAM file)**

OpenTP1 file accessible by the direct access method. A file can be referenced and modified using a relative block number as an access key to the data. The sequential access method can be used to reference data only.

### **domain**

A logical division of a network. Domain management uses the domain name service (DNS) or the network information service (NIS). When performing client/server communications in a large-scale network, you can improve the efficiency of scheduling by making a domain modification to remote procedure calls of OpenTP1.

### **DTP**

See *X/Open Distributed Transaction Processing (DTP) model*.

### **dynamic loading of service functions**

A facility that dynamically loads (reads) the service functions contained in a UAP shared library. A UAP shared library is created by linking the set of UAP object files generated by compiling the UAP source files. Use of this facility enables you to add and delete service functions simply by changing the `service` operand of the user service definition. There is no need to change the stub or re-create the UAP executable file. Because the service functions are loaded when the UAP starts, no stub or service functions are needed to create the UAP executable file.

### **global archive journal service**

A facility to collect system journals from OpenTP1 nodes that are used in a cluster system or parallel-processing system. This facility makes it unnecessary to unload system journals from each OpenTP1 node.

See also *archived-journals file (or archive journal file)*.

### **global cache**

An area used by the name service to manage service information for servers running at other nodes.

### **global transaction**

A UAP process that performs transaction processing is called a transaction branch. Use of RPCs enables you to perform transaction processing that consists of multiple UAP processes. A set of transaction branches that consists of multiple UAP processes is called a global transaction.

A transaction branch that starts a transaction is called a root transaction branch.

### **host**

A machine that is connected to a network and on which OpenTP1 runs. If the MultiOpenTP1 configuration is used, a host contains more than one OpenTP1



instance.

### **import**

An operation for registering an OpenTP1 scenario template definition file in JP1/AJS2 - Scenario Operation as a scenario template.

### **input queue**

An ordered group of MCF messages that came from other systems and that are waiting to be processed.

The TP1/Message Control product manages input queues.

### **inquiry-response message**

A message that requires a response. See also *send-only message*.

The TP1/Message Control product manages inquiry-response messages.

### **Internode Load-Balancing facility**

A facility that balances the load between nodes by placing user servers of the same service group on multiple nodes. When a service is requested, it is allocated to one of the user servers of the service group. The least-loaded node can be selected according to the scheduling of nodes.

### **internode shared table (IST)**

A data table managed by the IST service, a facility provided by the product TP1/Shared Table Access. When the IST service is used, the user should set the name of the table in the function of the IST service. Because the internode shared table is managed by the TP1/Shared Table Access of each system, the UAP does not need to manage the node location of the internode shared table.

### **journal**

Information about system operations. OpenTP1 obtains such information to check and record the operating status of the system, and to recover after a failure. The following journals can be obtained using OpenTP1:

- journals used for a complete or partial recovery
- system trace information
- user-specified journals

### **local cache**

An area used by the name service to manage service information for the servers running at the local OpenTP1 system.

### **logical message**

A unit for messages sent between two systems. A logical message contains a set of

related tasks, and consists of one or more segments.

The TP1/Message Control product manages logical messages.

### **logical terminal**

A logical name of a communication target in a remote system. The logical name is used when a UAP sends or receives messages.

The TP1/Message Control product manages logical names.

### **MCF**

See *message control facilities (MCF)*.

### **MCF main function**

A user-created main function for use with TP1/Message Control, an OpenTP1 resource manager. The `main` and `dc_mcf_svstart()` functions (described in the manual *OpenTP1 Operation*) are issued from the MCF main function. If a user-exit routine is included in the system, the routine's address must be set in the MCF main function. The MCF main function is written for a communication protocol, or for a service that starts applications. The `main` function for each MCF main function needs to be written independently.

The TP1/Message Control product manages MCF main functions.

### **MCF message queue**

There are two kinds of MCF message queues: an input queue that stores MCF messages received via TP1/Message Control, and an output queue that stores MCF messages to be sent to other systems.

The TP1/Message Control product manages MCF message queues.

### **message control facilities (MCF)**

The OpenTP1 facilities provided by the products TP1/Message Control, TP1/NET/Library, and the products that correspond to the relevant communication protocol. These products provide the MCF message-exchange capabilities of OpenTP1.

### **message handling program (MHP)**

One of the OpenTP1 UAPs that is used for MCF message communication. This program is made up of service functions that receive MCF messages from other systems and a main function that organizes the service functions.

The TP1/Message Control product manages message handling programs.

### **message log**

System management information output by OpenTP1. An MCF message log is displayed on a screen and is also written into a specified file. You can use the `logcat` command to edit and display the stored message log.

The message log of OpenTP1 can be reported to the application programs exclusively created in a system. The application program that has received a report can report the status of OpenTP1 to other vendors' network management systems. If a message log is reported, specify Y in the `log_notify_out` operand in the log service definition of OpenTP1.

**message queue**

See *MCF message queue* or *MQA message queue*.

**message queue access (MQA)**

The OpenTP1 facility provided by the product TP1/Message Queue. This product provides the MQA message-queuing capabilities of OpenTP1.

**message queue interface (MQI)**

An API consisting of instructions used by UAP when performing communication using message queuing. MQI is a standard API of the WebSphere MQ.

**message queuing**

See *MQA message queuing*.

**message sequential number**

A number assigned to each MCF message. These numbers are used to avoid resending the same MCF message or omitting messages.

The TP1/Message Control product manages message sequential numbers.

**MHP**

See *message handling program (MHP)*.

**MQA**

See *message queue access (MQA)*.

**MQA message queue**

A message-putting queue for communication using MQA message queuing. The messages put in an MQA message queue are sent by the queue manager. Depending on the MQA service definition and the argument of MQI, desired messages can be gotten regardless of the order in which they were put.

**MQA message queue file**

A file that stores the message queues for communication using MQA message queuing. For OpenTP1, a user should create the queue file in the OpenTP1 file system by using commands.

**MQA message queuing**

A message-queuing communication procedure developed by the IBM Corp. In

communication using message queuing, the queue manager puts or gets the messages queued by UAPs. Therefore, UAPs are free from processing inter-system communication and handling communication errors. Since UAPs can put messages at any time, they can be used like e-mail. Access from a UAP to a queue uses Message Queue Interface (MQI), the standard application programming interface of the WebSphere MQ.

### **MQA service**

OpenTP1 system's services necessary for communication using message queuing. The MQA service manages the message queues and UAP processes.

### **MQ system**

A system (node) where the queue manager of message queuing exists

### **MQT service (message queue transfer service)**

OpenTP1 system's services necessary for communication using message queuing. The MQT service manages inter-system communication by using TCP/IP protocol.

### **MultiOpenTP1**

Enables configurations in which two OpenTP1 instances are running on one host. Each OpenTP1 is operated independently so, for example, one OpenTP1 instance can be used for actual processing and the other can be used for testing new applications.

### **Multiserver facility**

A facility for handling more than one service request at the same time by processing the requests on more than one server process. The Multiserver facility improves the efficiency of UAP processing.

### **node**

A host or an OpenTP1 node. See also *host* and *OpenTP1 node*.

### **OpenTP1 administrator**

A UNIX user who administers the OpenTP1 system. An OpenTP1 administrator has the privileges required to perform important operations on OpenTP1. The user name for the OpenTP1 administrator is determined by a superuser. The user name must be password-protected.

### **OpenTP1 file system**

The file system provided by OpenTP1. Each file in an OpenTP1 file system is called an *OpenTP1 file*. Data requiring a high degree of reliability can be stored in this OpenTP1 file system: for example, important user data or system journals used at recovery.

**OpenTP1 home directory**

The directory that stores files and directories used by OpenTP1.

In OpenTP1, the OpenTP1 home directory is managed using the `DCDIR` environment variable. On a machine on which OpenTP1 is installed, you can move from any directory to the OpenTP1 home directory by specifying `$DCDIR`.

In OpenTP1 manuals, the keyword `$DCDIR/` is used to represent the OpenTP1 home directory.

**OpenTP1 installation directory**

The directory in which OpenTP1 is installed. The path of the OpenTP1 installation directory differs depending on the OS, as shown below.

AIX, HP-UX, and Solaris

`/BeTRAN`

Linux

`/opt/OpenTP1`

Windows

`C:\OpenTP1`

Note that the `C:\` part may be different depending on your environment.

**OpenTP1 node**

OpenTP1 systems that make up a multinode area or a multinode subarea. Each OpenTP1 node is distinguished by its node identifier which is specified in the system common definition. The two OpenTP1 systems in a System Exchange configuration are regarded as one OpenTP1 node.

**output queue**

An ordered group of MCF messages that are waiting to be sent to other systems.

The TP1/Message Control product manages output queues.

parallel-processing system

See *cluster system or parallel-processing system*.

**partial recovery**

The process in which, when an error occurs in a UAP, the transaction being executed is canceled and all the resources related to the transaction are returned to their previous states.

**process**

The in-memory processing created when user servers or OpenTP1 use OS memory. To

maintain performance OpenTP1 controls the number of processes.

### **queue group ID**

An identifier for a message queue file that is used for message exchange. This ID is used with OpenTP1 commands to display the status of a queue group.

The TP1/Message Control product manages queue group IDs.

### **queue manager**

A software product that manages MQA message queuing. A queue manager is required in a system that uses MQA message queuing to communicate. For OpenTP1, TP1/Message Queue serves as a queue manager.

### **real-time statistics**

The statistics that can be output in real time for the entire system, for each server, and for each service are called *real-time statistics*. By outputting real-time statistics, you can check the operating status of the OpenTP1 system in real time, and quickly perform system operation management and error recovery.

### **real-time statistics service**

An OpenTP1 facility for managing real-time statistics in order to check the operating status of the OpenTP1 system in real time.

### **remote procedure call (RPC)**

A method of communicating between processes that are executing UAPs. OpenTP1 UAPs use RPCs to communicate with UAPs on other systems. When using RPCs, programmers do not need to know the network address of the target UAP.

### **resource manager**

A generic name for facilities that manage resources. A DBMS is a resource manager.  
response message

See *inquiry-response message*.

### **rollback**

An operation at a synchronization point to cancel a transaction. A transaction can be rolled back in two ways: a user issues a function from an application program, or OpenTP1 stops the application that carries out the transaction.

### **root transaction branch**

A transaction branch, which is a UAP process, that belongs to a global transaction and is where the transaction starts.

### **RPC**

See *remote procedure call (RPC)*.

**RPC suppression list**

A list containing information about the OpenTP1 nodes that have not started. This list is owned by each OpenTP1 system.

**scenario**

An object that associates scenario templates with jobs to execute a sequence of jobs as defined. JP1/AJS2 - Scenario Operation registers scenario templates in JP1/AJS2 as a scenario, and JP1/AJS2 executes the scenario.

**scenario job**

An object that associates scenario job templates with jobs to execute a sequence of jobs as defined. JP1/AJS2 - Scenario Operation uses scenario jobs. A scenario job is an object for executing the commands, shell scripts, Windows executable files, and other items defined in a scenario.

**scenario job template**

A part of a scenario template used by JP1/AJS2 - Scenario Operation. A scenario job template defines the commands, shell scripts, Windows executable files, and other items defined in a scenario template. In OpenTP1, scenario job templates are used when the scenario template functionality of JP1/AJS2 - Scenario Operation is used.

**scenario library**

A folder for managing scenario templates in JP1/AJS2 - Scenario Operation.

**scenario template**

A component that defines a job sequence and that can be used as a boilerplate. JP1/AJS2 - Scenario Operation uses scenario templates. In OpenTP1, scenario templates are used when the scenario template functionality of JP1/AJS2 - Scenario Operation is used.

**scenario variable**

A variable that is preset in a scenario and that can be changed according to the operating environment. For example, information about OpenTP1 directories may be set in scenario variables.

**scheduler**

One of the OpenTP1 facilities that starts and stops user servers according to changes in the load on each node. This facility improves system performance by dispersing loads across different nodes and controlling the number of processes.

**send-only message**

An MCF message that does not require any response. You can assign priorities to each send-only message. See also *inquiry-response message*.

The TP1/Message Control product manages send-only messages.

### **server gateway program (SGW)**

A gateway program that enables OpenTP1 to access other open systems. An SGW receives requests for services from an application in an open system other than OpenTP1, and it requests the same service from OpenTP1 service-providing programs. The `dc_rpc_call()` function provided by TP1/Client is used to request a service. In an OpenTP1 system a server gateway program is regarded as a client user program.

### **server UAP**

See *client UAP/server UAP*.

### **service**

Procedure required to be carried out for clients in a client/server system. In C, a service is coded as C functions and the completed functions are called service functions. In COBOL, a service is coded as a subroutine and is called a service program.

### **service group**

An OpenTP1 server UAP that provides a set of services to process requests from clients. To request a service, a service group name and a service name are specified as arguments of `dc_rpc_call()`. The OpenTP1 `dc_rpc_call()` function is used for making remote procedure calls in OpenTP1 systems.

### **service information prioritizing function**

A function for returning service information for a specific node preferentially when the name service returns service information to the client UAP that issued a service request. The node chosen by this function for return of service information is called the *priority selection node*.

This function enables you to normally use the server UAP at the priority selection node and to use a different node only in the event of a failure. You can treat one server UAP as the running system and another as a standby system.

### **service-providing program (SPP)**

An OpenTP1 UAP that can perform as a server. This program consists of service functions that provide the service requested by a client UAP, and a main function to organize the service functions.

### **service-using program (SUP)**

An OpenTP1 UAP designed to perform as a client only. It does not contain any functions to provide services for other UAPs and it only requests services from a service-providing program.



**SPP**

See *service-providing program (SPP)*.

**status service**

An OpenTP1 facility that manages system information, such as the operation status of UAPs.

**Structured Transaction Definition Language (STDL)**

A high-level language used by a UAP when performing distributed transactional communication that conforms with the Multivendor Integration Architecture (MIA) standard. The STDL specification is defined by the MIA consortium. A program coded using STDL is known as an *STDL task*.

**stub**

A program that functions as a library to support RPCs in client/server communications, which connects requests for services with the services provided by server UAPs. The `stbmake` command or `tpstbmk` command creates a stub from a RPC interface definition file that is made by a user. The stub is created as a C program. To link the stub to an executable UAP file, the stub must be compiled. The following UAPs need stubs: service-providing programs when an RPC or the XATMI interface is used, service-using programs when the XATMI interface is used, and message-handling programs. However, no stub is required if all the service functions are rolled into a UAP shared library and are loaded dynamically. A UAP shared library is created by linking the set of UAP object files generated by compiling the UAP source files.

For more details on stubs, see the *OpenTP1 Programming Guide*.

**SUP**

See *service-using program (SUP)*.

**superuser**

A user with UNIX OS privilege. A superuser has access permission to all files in the UNIX file system. The user login name of a superuser is `root`.

**synchronization point**

A breakpoint between transactions. In a commit operation on a transaction; the effects of a transaction up to a synchronization point are implemented. In a rollback operation on a transaction, a transaction cannot be completed and the status of resources is returned to the same status as at the previous synchronization point.

**system service**

Another name for an OpenTP1 facility; a *user server* is another name for an OpenTP1 UAP.

table access method file (TAM file)

A file accessed by using a simple structure table that is specially created for use in OpenTP1. A table name and a key value are used to access the simple structure table. Accessing the table enables fast access for such tasks as retrieving, cancellation of retrieving, modifying, adding and deleting of records, and obtaining table information.

## **TAM**

See *table access method file (TAM file)*.

## **TCP/IP (Transmission Control Protocol/Internet Protocol)**

A protocol developed by ARPANET in the Defense Advanced Research Project Agency (DARPA) project. The TCP/IP protocol is mainly used for LAN.

## **TP monitor**

See *transaction processing monitor (TP monitor)*.

## **transaction**

A logical unit of operation that is regarded as a single operation. In transaction processing, each transaction ends in complete success or complete failure. For example, after a transaction, a *commit* operation ensures that all the file modifications required by a transaction are completed, or a *rollback* operation ensures that none of the file modifications are completed.

## **transaction branch**

Each UAP process that is part of a global transaction. The transaction branch in which the global transaction starts is called a root transaction branch. See also *global transaction* and *root transaction branch*.

## **transactional RPC**

An RPC that OpenTP1 handles as a transaction. For example, coding `dc_trn_begin()` before `dc_rpc_call()` will cause OpenTP1 to handle the `dc_rpc_call()` call as a transaction: the `dc_rpc_call()` is the transactional RPC. To become part of the transaction of a transactional RPC, a UAP process that is requested for a service must have the transaction attribute (`atomic_update = Y`) specified in the user service definition.

## **transaction manager**

The OpenTP1 facility that manages and executes transactions.

transaction processing monitor (TP monitor)

Software that monitors and controls transactions. This software provides the infrastructure for constructing an online system. Major features include the communication facility for transferring data among terminals and other machines, and the recovery facility to prevent loss or mismatch of data in case of a failure.

**trigger facility**

A facility that notifies the local system's UAP that a message has arrived at the message queue when performing communication using MQA message queuing. The UAP assigned to receive trigger events is called a *trigger monitor application*.

**UAP**

See *user application program (UAP)*.

**uptime statistics**

Information about the operation of OpenTP1 system services and user servers. Uptime statistics are used to monitor the operational status of the entire OpenTP1 system.

**user application program (UAP)**

A program that carries out user tasks. An OpenTP1 user server is an example of a UAP. In non-OpenTP1 systems, a UAP is often called an *application program*.

**user exit routine**

A user-coded program for making a message-exchange UAP applicable to a wider range of tasks. For example, a user exit routine can be used to determine what application to launch, or to edit a message before the UAP sends it to another system.

**user journal**

Any information specified by a user to be stored in a system journal file. User journals are collected during UAP processing.

**user server**

A generic name for processes that execute OpenTP1 UAPs that function as user-created servers in OpenTP1 systems, or for those OpenTP1 UAPs themselves.

**WebSphere MQ**

A generic name of the products for MQA message-queuing-type communication developed by the IBM Corp. OpenTP1 allows message-queuing-type communication, using TP1/Message Queue.

**X/Open Distributed Transaction Processing (DTP) model**

A distributed processing system model defined by X/Open: an organization for the standardization of open systems. The DTP model consists of a transaction manager that manages and executes transactions; a resource manager, which manages system resources; and application programs.



---

# Index

---

## A

abbreviations for products iv  
abnormal termination 142  
    from status file error 247  
    no swappable file 266  
    recovering from UAP 325  
access mode for TAM table 297  
acronyms ix  
ans-type 123  
Application Activate facility 127, 140  
Application Program 17  
application startup 127  
application startup process 128  
archive journal file 267  
    available status of 270  
    available/current status of 270  
    available/standby status of 270  
    creating 268  
    duplicating 269  
    element file name in 268  
    filegroup name in 268  
    numbers of parallel accesses 269  
    parallel access facility for 268  
    physical file name in 268  
    purpose of 267  
    recovering from error 328  
    status of 270  
    structure of 267  
    unavailable status of 270  
    unloading 271, 370  
    with Global Archive Journal facility 367  
archive-journal source node 367  
archive-journal target node 367  
association 190  
asynchronous-response RPC 84  
automatic connection mode 202  
automatic system switchover 350

## B

backing up  
    DAM and TAM 318  
    DAM file online 280  
    OpenTP1 file system 243  
balancing load  
    among nodes 166  
    with Multiserver facility 163  
block in deferred update 279  
block length extension facility 280  
branch, specifying 67  
buffer area, saving shared memory in 177

## C

C function for DAM file 278  
C++ 7, 44  
cache block 283  
cache block securing 283  
cacheless access 293  
called 190  
calling 190  
chained RPC 84  
changing  
    definitions 318  
    file capacity 319  
    network configuration while online 319  
    replacing UAPs online 319  
    SPP online 149  
checkpoint 259  
checkpoint dump file 259  
    adding physical file for 261  
    duplicated 259  
    fallback facility for 263  
    operation when, is duplicated 264  
    purpose of 259  
    recovering from error 327  
    structure of 259  
clean-up processing 283

## Index

- client
    - service-using program used as 46
    - software, using 32
  - Client .NET 75
  - client facility 182
  - client gateway program 45
  - client service 185
    - definition of 185
  - client UAP 31
  - client user program 45
  - client/server configuration 28, 81
    - communication protocol 31
  - CLTIN 182
  - CLTOUT 182
  - cluster system with Multinode facility 363
  - command used to start UAP 133
  - commit 60, 62
    - heuristic 66
    - heuristic decision 65
    - two-phase 63
  - commit processing 64
  - communication
    - in client/server configuration 81
    - via RPC 81
    - via RPC that uses TxRPC interface 116
    - via RPC that uses XATMI interface 113
    - with TP1/Client 32
    - with XDM/DCCM3 188
  - communication functions
    - asynchronous 122
    - synchronous 122
  - communication protocol
    - client/server configuration 31
    - product 398
  - communication resource manager 16
  - compatibility with ORACLE 307
  - complete recovery 321
    - opening reserved filegroup at 258
    - service recovery in 322
    - system recovery in 321
    - transaction recovery in 322
  - configuration example 8
  - connection 136
  - connection mode 202
  - Connector .NET 76
  - cont-type 124
  - continuous inquiry response type 124
  - conventions
    - abbreviations for products iv
    - acronyms ix
    - diagrams xii
    - fonts and symbols xiii
    - KB, MB, GB, and TB xiv
    - version numbers xiv
  - Cosminexus TP1 Connector 76
  - creating
    - archive journal file 268
    - checkpoint dump file 240
    - DAM file 240, 278
    - MCF message queue file 240
    - MQA message queue file 240
    - OpenTP1 file system 240
    - server recovery journal file 267
    - status file 240
    - system journal file 240
    - TAM file 240, 297
  - CRM 17
  - CUP 45
  - current status of
    - archive journal file 270
    - status file 247
    - system journal file 256, 271
- ## D
- DAM 13
  - DAM file
    - access to unrecoverable 282
    - compared to TAM file 294
    - creating 278
    - deadlock in 299
    - deferred update of 279
    - description of 276
    - FRC (file recovery) facility 243
    - I/O function for 278
    - lock on 278
    - offline backup of 280
    - online backup of 280
    - recovering from error 329

- dambkup 280
  - damfrc 328
  - damhold 280
  - damload 241, 278
  - damrm 280
  - data operations using extended presentation facility 35
  - database management system, accessing 307
  - DBMS 307
  - dc\_clt\_cltin\_s() 182
  - dc\_clt\_cltout\_s() 182
  - dc\_dam\_create() 278
  - dc\_dam\_put() 278
  - dc\_jnl\_ujput() 256
  - dc\_lck\_release\_all() 212
  - dc\_lck\_release\_byname() 212
  - dc\_mcf\_execap() 127, 140
  - dc\_mcf\_receive() 34, 137
  - dc\_mcf\_recvsync 125
  - dc\_mcf\_reply() 34
  - dc\_mcf\_rollback() 127
  - dc\_mcf\_send function 124
  - dc\_mcf\_send() 138
  - dc\_mcf\_sendrecv 125
  - dc\_mcf\_sendsync 125
  - dc\_rpc\_call() 82
  - dc\_rpc\_cltsend() 186
  - dc\_rpc\_set\_service\_prio() 147
  - dc\_tam\_close() 298
  - dc\_tam\_open() 298
  - dc\_tam\_read() 298, 299
  - dc\_trn\_begin() 31, 68, 69, 308
  - dc\_trn\_chained\_commit() 68
  - dc\_trn\_chained\_rollback() 63, 68
  - dc\_trn\_unchained\_commit() 308
  - dc\_trn\_unchained\_rollback() 63
  - dclog1 218
  - dclog2 218
  - dcmstart 366
  - dcmstop 366
  - dcndls 366
  - dcreset 318
  - dcsetup 315, 318
  - dcshmls 384
  - dcstart 317, 357
  - dcstop 317, 360, 361
  - dcsvstart 46
  - deadlock 214, 299
    - recovering from 326
  - debugging 55
  - deferred update of DAM file 279
  - definitions
    - changing 318
    - necessary for using remote API facility 203
    - with DBMS 309
  - delvcmd 373
  - diagram conventions xii
  - disk
    - mirroring 356
    - partitioning for file system 234
  - disk queue 134
    - processing when, is full 136
  - distributed transaction processing 60
  - distributed transactions 60
  - distributions
    - maximum number of 269
    - minimum number of 269
  - domain name system 86
  - DTP model 16
    - configuration of 17
  - uplicated system 350
  - uplicating
    - archive journal file 269
    - checkpoint dump file 264
    - journals 216
  - dynamic connection schedule mode 201
- E**
- element file name 268
  - entity, terminal 138
  - environment
    - setting up 312
    - setup tasks performed by OpenTP1 administrator 315
    - setup tasks performed by superuser 314
    - variables, setting 309
  - error recovery 321
  - EX lock mode 211

## Index

- example of
  - possible configuration 8
  - process control with Multiserver facility 179
  - using remote API facility 199
- extended internode load-balancing facility 172
- extended presentation facility 35
  - Microsoft Windows with 35
  - X/P system with 35
  - X/W system with 35
- extended RM registration definition 308

## F

- facilities
  - application activate 127, 140
  - DAM FRC (file recovery) 243
  - extended presentation 35
  - file recovery journal integration 217
  - for monitoring node registered in RPC
  - suppression list 112
  - for obtaining message log 217
  - global archive journal 367
  - integrated system management 41
  - journal duplicating 216
  - journal editing 217
  - journal maintenance 216
  - locking 211
  - multinode 363
  - multiserver 55, 84, 162
  - statistics output 222
  - System Switchover 350
  - testing and debugging 55
- fail-safe system 350
- failure recovery 321
- fallback facility
  - for checkpoint dump 263
  - guaranteed-valid generation when using 263
- fallback using memory queue 136
- FIFO
  - when scheduling messages to MHP 157
  - when scheduling service requests 147
- FIL event trace 347
- FIL event trace information file 347
- filchmod 244
- filchown 244

- file capacity, changing 319
- file error 327
- file recovery journal integrating facility 217
- file recovery journal, integrating 216
- file system 233
  - assigning 244
  - backing up/restoring 243
- filegroup
  - in archive journal file 267
  - in checkpoint dump file 259
  - in status file 246
  - in system journal file 249
- filegroup name 268
- files
  - archive journal 367
  - changing capacity of 319
  - creating OpenTP1 file system 240
  - DAM file 276
  - element file name 268
  - message log 218
  - physical file name 268
  - status 246
  - system journal 249
  - TAM file 293
- filmkfs 315
- font conventions xiii
- forced normal termination 317
- forced termination 142
- FRC (file recovery) facility 243
- front-end processor 9

## G

- GB meaning xiv
- generation, guaranteed-valid 262
- Global Archive Journal facility 267, 367
- global archive unloaded-journals file 370
- global cache 458
- global search facility 90
- global transaction 62
- grouped system switchover 350
- guaranteed-valid generation 262
  - when using fallback facility 263



**H**

HAmonitor 16, 353  
 heuristic decision 65  
     heuristic mix 66  
     processing in 67  
     transaction determination 65  
 heuristic hazard 65  
 heuristic mix 66  
 hold message 136  
 host, multi-homed 374  
 hot-standby system 350

**I**

I/O function for DAM file 278  
 IDL compiler with TxRPC interface 116  
 IDL-only TxRPC 116  
 infinite loop, recovering from 324  
 input function for DAM file 278  
 input queue 34, 134  
     description of 134  
 inquiry-response communication 123  
 integrated journal file 216  
 integrated system management facility 41  
 interface definition language file 117  
 interfaces among UAP, TM, RM, and CRM 16  
 internal channel 127  
 internode load-balancing facility 166, 167  
 internode shared table 301  
     access environment for 303  
     accessing 303  
     overview of 13  
     structure of 303  
 IP address, correspondence between OpenTP1  
 instance and 374  
 ISAM file 16, 306  
     recovering from error 329  
 IST service 301  
     configuration of 301

**J**

J2EE Connector Architecture 73  
 JNL performance verification trace 338

JNL performance verification trace information  
 file 338  
 jnlcolc 216  
 jnldcopy 216  
 jnldelpf 261  
 jnledit 216  
 jnlinit 240, 315  
 jnlls 268  
 jnlmct 216  
 jnlmkrf 267, 328  
 jnlopnfg 268  
 jnlrput 217  
 jnlstts 216  
 jnlunlfg 259, 267, 271, 318  
 journal block 264  
 journal duplicating facility 216  
 journal editing facility 217  
 journal maintenance facility 216  
 journals  
     editing 217  
     integrating file recovery 216  
     recovery journal 254  
     size of 264  
     statistical journal 255  
     synchronization point journal 254  
     user journal 256

**K**

KB meaning xiv

**L**

LAN  
     component to be used 356  
     maintenance 353, 356, 364  
     monitoring 353  
     that uses client/server processing 8  
 language for message log, specifying 219  
 large-scale computer 10  
 LCK performance verification trace 339  
 LCK performance verification trace information  
 file 339  
 lckrminf 215  
 link  
     monitoring 353

## Index

- with DBMS 309
- load balancing
  - among nodes 166
  - with Multiserver facility 163
- loading opportunity 298
- local cache 459
- local memory 384
- lock monitoring time 299
- lock on TAM table 299
- locking
  - deadlock 214
  - EX mode 211
  - on DAM file 278
  - PR mode 211
  - releasing 212
  - resource 211
  - specifying whether to wait 212
  - units 211
- logcat 218
- logical channel 137
- logical file
  - in DAM file 276
  - in TAM file 293
- logical filegroup
  - in archive journal file 267
  - in checkpoint dump file 259
  - in status file 246
  - in system journal file 249
  - unload check on 257
- logical message 121
- logical terminal 138
- logical terminal name 138

**M**

- mainframe 10
- maintenance LAN 353, 356, 364
- maximum dispersed files for parallel access 254
- MB meaning xiv
- MCF
  - applications in MHP scheduling 157
  - starting 126
- MCF application 51
- MCF application definition 121
- MCF communication events 128

- MCF communication process 128, 137
- MCF error events 128
- MCF events 128
- MCF message exchange 120, 122
  - configuration of 34
  - configuration using extended presentation facility 35
  - description of 34
  - input queue 34
  - network that can use 35
  - output queue 34
  - overview of 34
  - supported protocols for 398
  - using TCP/IP protocol 187
- MCF message queue 34, 134
- MCF message queue file 137, 138, 273
  - purpose of 273
  - recovering from error 329
  - structure of 273
- MCF online tester 56
- MCF performance verification trace 341
- MCF system statistics 222
- MCF trace 332
- mcfadctap 158
- mcfstdctv 159
- mcfstdctsg 158, 160
- mcfthldiq 135
- mcfthldoq 135
- mcftrlsiq 135
- mcfuevt 133
- memory
  - local 384
  - shared 384
  - structure of 384
- memory queue 134
  - fallback using 136
- message control 120
- message control facility 13
- message exchange 34
  - screen data operations using 35
- message exchange facility 34
- message log
  - obtaining 217
  - specifying language for output 219

- suppressing 219
  - message log file 218
    - recovering 328
  - message queue 38
  - message queue access 14
  - message queue file 329
  - message queue interface 39
  - message sequence numbers 219
  - message sequential number 139
  - message-handling program 34, 51
  - message-storing buffer pool 177
  - messages
    - getting 40
    - handing, when OpenTP1 terminates 141
    - hold 136
    - order of sending 143
    - putting 39
    - receiving 120, 136
    - sending 120, 138
    - sending and receiving 134
    - sequence numbers of 219
    - unprocessed 142
  - MHP 34, 51, 52, 120
    - automatic shutdown of scheduling 161
    - client user program 45
    - overview of 45
    - scheduling requests to 156
    - shutdown scheduling for 157
    - starting, by issuing UAP function 127
    - starting, by MCF event 128
    - transaction 69
  - minimum dispersed files for parallel access 254
  - mirrored disk 356
  - modifying 319
  - monitor 330
  - monitored RM definition 221
  - monitoring
    - output queue 141
    - status 320
    - temporary closing request 392
    - to detect infinite loop 324
    - TP monitor 7
  - monitoring LAN 353
  - monitoring link 353
  - monitoring path 353
  - monsbystp 361
  - monswap 361
  - MQA message
    - getting 40
    - putting 39
  - MQA message queue 38, 39
    - queue manager 38
    - transaction processing 40
  - MQA message queue file 274
    - description of 274
    - purpose of 274
    - recovering from error 329
  - MQA message queuing 274
    - queue manager 28
  - mqainit 240
  - MQI 39
  - MSDTC linkage 71
  - multi-homed host 374
  - multi-node area 365
  - multi-node subarea 366
  - multi-scheduler facility 175
  - Multigeneration Guarantee facility 262
    - recovering, when using 262
  - Multinode facility 363
    - available operations 366
    - Global Archive Journal facility 367
    - overview of 363
  - MultiOpenTP1
    - commands, distributing 372
    - configuration of 372
    - description of 372
    - using 320
  - multiple instances, using 349
  - Multiserver facility
    - balancing load in 163
    - overview of 162
    - process control with 179
  - multiserver facility 55
    - drawback 84
- N**
- NAM event trace 346
  - NAM event trace information file 346

- namdomainsetup 89
  - name service 82
  - namndchg 319
  - network configuration, changing 319
  - network error, recovering from 329
  - network that can use MCF message exchange 35
  - network transparency 82
  - no-wait RPC 84
  - noans-type 124
  - node
    - balancing loads among nodes 166
    - containing two OpenTP1 instances 372
    - internode locking 211
    - OpenTP1 366
    - with Global Archive Journal facility 267
  - node management 81, 105
  - node monitoring facility 108
  - non-automatic connection mode 202
  - non-resident process 163
  - non-response 124
  - normal termination 142
  - number of ports, restricting 390
  - numbered messages, requesting 219
- O**
- offline tester 55
  - one-system operation
    - for status file 247
    - for system journal file 250
    - of checkpoint dump file 264
  - online
    - backing up DAM file 280
    - changing network configuration while 319
    - changing SPP 149
  - online tester 56
  - OpenTP1 administrator
    - operation tasks performed by 317
    - tasks for setup 312
  - OpenTP1 file 234
    - protecting 243, 244
  - OpenTP1 file system 234
    - assigning 244
    - backing up 243
    - difference from OS file system 236
    - restoring 243
    - selection of files to create 238
  - OpenTP1 instance 372
    - correspondence between IP address and 374
  - OpenTP1 node 366
  - OpenTP1 process, recovering 243
  - OpenTP1 services 20
    - types of 20
  - OpenTP1 software products 12
    - list of 12
  - OpenTP1 system
    - operating 317
    - setting up 312
  - OpenTP1 system definition 23
  - OpenTP1 system service 20
  - OpenTP1 termination mode 142
  - operation 317
    - routine 317
    - with System Switchover facility 357
  - operations by users
    - for backing up DAM and TAM 318
    - for changing file capacity 319
    - for modifying definitions 318
    - for outputting message log to file 318
    - for replacing UAP 319
    - for starting OpenTP1 317
    - for starting UAP 318
    - for terminating UAP 318
    - for unloading journal 318
  - ORACLE 307
  - OS file system, difference from OpenTP1 file system 236
  - output function for DAM file 278
  - output queue 34, 134
    - description of 134
    - monitoring message in 141
  - outputting
    - message log to file 318
    - statistics 222
  - overwrite check 262
- P**
- parallel access facility 268
    - for system journal file 251

- parallel accesses guaranteed, minimum number of 269
  - parallel processing
    - of services 162
    - with Multinode facility 363
  - partial recovery 321, 324
    - of transaction 326
    - using system journal file 249
  - performance verification trace 334
  - permanent connection 201
  - physical file
    - adding, for checkpoint dump file 261
    - in archive journal file 267
    - in checkpoint dump file 259
    - in DAM file 276
    - in MCF message queue file 273
    - in status file 246
    - in system journal file 249
    - in TAM file 293
    - operating with one, in status file 247
    - operating with one, in system journal file 250
  - physical file name 268
  - planned system switchover 350
  - planned termination 142
  - PR lock mode 211
  - prepare processing 64
  - primary system 351
  - priority selection node 93
  - process
    - control 162
    - control, example of 179
    - resident and non-resident 163
  - process service event trace 347
  - process service event trace information file 347
  - protecting file system 243
  - protocol product, system connected to 399
  - purpose of
    - archive journal file 267
    - checkpoint dump file 259
    - MCF message queue file 273
    - message queue file 273
    - transaction recovery journal file 266
    - server recovery journal file 267
    - status file 246
    - system journal file 249
- Q**
- queinit 240
  - queue
    - schedule queue for MHP 156
    - schedule queue for SPP 147
    - server that receives request from 152
  - queue file 273
    - dc\_mcf\_execap() 140
  - queue group 273
  - queue manager 28, 38, 40
- R**
- rap-processing client manager 203
  - real-time statistics service 22, 224
  - received message lifetime 142
  - received messages 137
  - receiving message 136
  - record-based lock 299
  - recovering 321
    - archive journal file 328
    - checkpoint dump file 327
    - DAM file 329
    - from deadlock 326
    - from error in transaction recovery journal file 266
    - from inability to start UAP 324
    - from infinite loop 324
    - from network error 329
    - from system failure 321
    - from UAP abnormal termination 325
    - from UAP failure 321, 324
    - ISAM file 329
    - MCF message queue file 329
    - message log file 328
    - MQA message queue file 329
    - server recovery journal file 267, 328
    - status file 327
    - system journal file 327
    - TAM file 329
    - transaction partially 326
    - transaction recovery journal file 328
    - using Multigeneration Guarantee facility 262

- using system journal file 249
  - recovery journal, contents of 254
  - reference-only TAM table 298
  - regular file 238
  - releasing
    - lock 212
    - scheduling shutdown by command 149
  - remote API facility 74, 196
  - remote procedure call 29
    - compressing send data for 86
    - of TP1/Client 183
  - replacing UAP 319
  - reporting
    - communication event failure 132
    - server startup to TP1/Client 185
  - requests, setting scheduling priorities for service 147
  - reserved filegroup, opening 258
  - reserved status of
    - archive journal file 270
    - status file 247
    - system journal file 256, 271
  - resident process 163
  - resource
    - consistency in transaction processing 60
    - locking 211
    - waiting to use 212
  - resource group 268, 369
  - resource manager 16
  - resource manager monitor service 220
  - response-type 123
  - restart 321
    - in complete recovery 321
  - restoring 321
  - return 34
  - RI 71
  - RM 17
  - rollback 60, 62
    - heuristic 66
  - rolling update 228
  - root transaction branch 62
  - round-robin scheduling 218
  - RPC 29
    - asynchronous-response 84
    - chaining 84
    - compressing data 86
    - no-response 82
    - non-response 84
    - relationship with SPP 50
    - response 82
    - synchronous-response 83
    - used to request service 82
    - using OpenTP1 library 81
  - RPC trace 333
  - RPC TxRPC 116
  - rpcdump 333
  - rpcmrg 334
  - rts 22
  - RTSSPP 224
  - RTSSUP 224
- S**
- scale in 227
  - scale out 227
  - scdhold 149
  - scdrls 149
  - scenario
    - rolling update 228
    - scale in 227
    - scale out 227
  - scenario template for system operation, using 227
  - schedule buffer group name 177
  - scheduling
    - automatic shutdown 161
    - priority for user servers 165
    - queue for MHP 147, 156
    - release scheduling shutdown 149
    - requests to MHP 156
    - requests to SPP 147
    - requests to UAP 147
    - setting priority for MHP 156
    - setting priority for SPP 147
    - shutdown for MHP 157
    - shutdown for SPP 148, 149
  - scheduling facility 147
  - secondary system 351
  - segment 121
    - in logical message 121
  - segment-receiving function 137

- send data, compressing 86
- send message lifetime 141
- sending messages 138
- sending priority 144
- server
  - scheduling priority for user 165
  - service-providing program 46
  - that receives request from queue 152
  - that receives request from socket 151
- server recovery journal file 267
  - creating 267
  - purpose of 267
  - recovering from error 328
  - recovering from error in 267
- server UAP 31
- service
  - executing in parallel 162
  - group in MHP scheduling 157
  - how RPC is used to request 82
  - in MHP scheduling 157
  - requests, setting scheduling priorities for 147
- service group 82
- service information prioritizing function 93, 466
  - notes about using 105
- service information searches, optional function for 90
- service recovery 322
- service request method 87
- service requests, setting scheduling priority for 147
- service-group name 50
- service-providing program 46
- service-using program 45, 46
- setting
  - OpenTP1 system 312
  - scheduling priorities for service requests 147
- setup, use, and error recovery, overview of 311
- SEWB3 43
- SGW
  - client user program 45
  - overview of 45
- shared memory 384
- shutdown
  - by command 149
  - of scheduling automatically 161
  - of scheduling for MHP 157
  - of scheduling for SPP 148
  - release of 149
- simple structure table 294
- single
  - physical file in status file 247
  - physical file in system journal file 250
- single-system operation 269
- socket, server that receives request from 151
- software products
  - HAMonitor 16
  - ISAM 16
  - Job Management Partner 1 41
  - overview of 12
  - SEWB3 43
  - TP1/Client/P 15
  - TP1/Client/W 15
  - TP1/FS/Direct Access 13
  - TP1/FS/Table Access 13
  - TP1/High Availability 14
  - TP1/LiNK 15
  - TP1/Message Control 13
  - TP1/Message Control/Tester 15
  - TP1/Message Queue 14, 38, 274
  - TP1/Multi 15
  - TP1/NET/HDLC 398
  - TP1/NET/High Availability 14
  - TP1/NET/HNA-NIF 398
  - TP1/NET/Library 14
  - TP1/NET/OSI-TP 398
  - TP1/NET/TCP/IP 398
  - TP1/NET/X25 398
  - TP1/NET/XMAP3 398
  - TP1/Offline Tester 14
  - TP1/Online Tester 15
  - TP1/Resource Manager Monitor 16
  - TP1/Server Base 13
  - TP1/Shared Table Access 13
- SPP 46
  - client user program 45
  - overview of 45
  - release scheduling shutdown 149
  - scheduling requests to 147
  - setting scheduling priority 147
  - shutdown scheduling for 148

## Index

- starting, by issuing UAP function 127
- that can use Multiserver facility 163
- transaction 68
- standby status of
  - archive journal file 270
  - status file 247
  - system journal file 256, 271
- standby system
  - starting, to complete postprocessing of running system 356
  - starting, to replace running system 356
- starting
  - OpenTP1 317
  - OpenTP1 with Multinode facility 366
  - OpenTP1 with System Switchover facility 357
  - recovering from inability to start UAP 324
  - transaction 68
  - UAP 318
  - UAP by issuing function 127
  - UAP by MCF event 128
  - UAP by using command 133
- startup notification facility 106
- static connection schedule mode 201
- statistical journal information 222
- statistical journal, contents of 255
- statistics output facility 222
- status
  - archived/not archived 371
  - available 256, 270
  - available/current 256, 270
  - available/standby 256, 270
  - monitoring output 320
  - of archive journal file 270
  - of OpenTP1 node 366
  - of status file 247
  - of system journal file 256, 271, 371
  - of TAM table 298
  - reserved 256
  - unavailable 256, 270
- status file
  - description of 246
  - operating with one physical file in 247
  - purpose of 246
  - recovering from error 327
  - shutdown status of 247
  - status of 247
  - structure of 246
  - swapping 248
- stbmake 52
- structure of
  - archive journal 267
  - checkpoint dump file 259
  - MCF message queue file 273
  - memory 384
  - processes 378
  - status file 246
  - system journal file 249
- stsinit 240, 315
- stsrn 248
- stub 47
- SUP 46
  - transaction 68
- superuser 312
- suppressing message log 219
- swappable file 258
- swapping
  - status files 248
  - system journal files 258
- symbol conventions xiii
- synchronization point 62
  - setting with MHP 69
  - setting with SUP or SPP 68
- synchronization point journal, contents of 254
- synchronous communication functions
  - receive 125
  - send 125
  - send/receive 125
- synchronous-response RPC 83
- system configuration using JP1 41
- system control information 321
- system definition 23
  - editing 23
- system journal file
  - available status of 256
  - available/current status of 256
  - available/standby status of 256
  - checking 262



- copying 257
  - description of 249
  - operating with one physical file in 250
  - purpose of 249
  - recovering from error 327
  - reserved status of 256
  - status of 256, 271
  - structure of 249
  - swapping 258
  - unavailable status of 256
  - unloading 257
  - system monitoring using audit logs 229
  - system recovery 321
  - system service 20
    - process 378
  - system service definition 23
  - system switching, procedure for 354
  - System Switchover facility 350, 353
    - monitoring LAN 353
    - monitoring link 353
    - operating with 357
    - overview of 350
  - system-switch system 350
- T**
- table-based lock 299
  - TAM 13
  - TAM data file 297
  - TAM file 294
    - compared to DAM file 294
    - creating 297
    - data file 297
    - deadlock in 299
    - description of 293
    - loading 298
    - online backup for 300
    - recovering from error 329
    - unloading 298
  - TAM table 293, 298
    - access mode for 297
    - internal functioning of 295
    - lock on 299
    - updatable but permits addition and deletion 298
    - updatable but prohibits addition and deletion 298
  - tamadd 295
  - tambkup 295
  - tamcre 241, 297
  - tamdel 295
  - tamfrc 328
  - tamload 298
  - tamrles 295
  - tamrm 295
  - tamunload 298
  - tasks
    - for backing up DAM and TAM 318
    - for modifying definition 318
    - for outputting messages log to file 318
    - for replacing UAP 319
    - for routine OpenTP1 operation 317
    - for starting OpenTP1 317
    - for starting UAP 318
    - for terminating UAP 318
    - for unloading journal 318
    - performed by OpenTP1 administrator 312
    - performed by superuser 312
  - TB meaning xiv
  - TCP/IP
    - protocol 35
  - TCP\_NODELAY 31
  - temporary closing 390
    - checking execution status of 392
    - information obtainable using command to check execution status of 393
    - request, monitoring 392
  - terminal entity 138
  - terminating
    - modes 142
    - OpenTP1 with Multinode facility 366
    - OpenTP1 with System Switchover facility 360
    - recovering from UAP abnormal termination 325
    - UAP 318
  - termination
    - normal 142
    - planned 142

## Index

- testing
    - UAP 55
    - with MultiOpenTP1 320
  - TP monitor 7
  - TP1/Client
    - facilities that need to be defined 185
    - preparing to request service from 182
  - TP1/Client communication 182
  - TP1/Client/J 15, 75
  - TP1/Client/P 15, 32
  - TP1/Client/W 15, 32
  - TP1/FS/Direct Access 13
    - DAM file 276
  - TP1/FS/Table Access 13
    - TAM file 293
  - TP1/High Availability 14, 353
  - TP1/LiNK 15
  - TP1/Message Control 13, 120
    - MCF online tester 56
  - TP1/Message Control/Tester 15
  - TP1/Message Queue 14, 38, 274
  - TP1/Multi 15, 364
  - TP1/NET/HDLC 398
  - TP1/NET/High Availability 14, 353
  - TP1/NET/HNA-NIF 398
  - TP1/NET/Library 14, 120
  - TP1/NET/OSI-TP 398
  - TP1/NET/TCP/IP 398
  - TP1/NET/X25 398
  - TP1/NET/XMAP3 35, 398
  - TP1/Offline Tester 14, 55
  - TP1/Online Tester 15, 56
  - TP1/Resource Manager Monitor 16, 220
  - TP1/Server Base 13
    - service-using program 45
  - TP1/Shared Table Access 13, 301
  - transaction
    - attribute 68
    - branch 62
    - branch, specifying 67
    - commit and rollback 62
    - determining 62
    - processing in MQA message queuing 40
    - root transaction branch 62
    - TP1/Client 185
    - UAP 67
    - with MHP 69
    - with SPP 68
    - with SUP 68
  - transaction manager 16, 17
  - transaction partial recovery 326
  - transaction processing, relation to 307
  - transaction recovery 322
  - transaction recovery journal file 266
    - purpose of 266
    - recovering from error 328
    - recovering from error in 266
  - trigger event 40
  - trigger facility 40
  - trigger monitor application 40
  - trncmt 65
  - trnftg 65
  - trnlkrm 308, 355
  - trnls 66
  - trnmkobj 309
  - trnrbk 65
  - trnrmid 309
  - trnstring 309
  - two-phase commit 63
  - tx\_begin() 308
  - tx\_commit() 308
  - TxRPC communication, creating UAP for 117
  - TxRPC interface
    - communication mode 116
    - communication via RPC that uses 116
  - typed buffer 116
  - typed record 116
- ## U
- UAP
    - changing online 319
    - creating, for TxRPC communication 117
    - failure, recovering from 324
    - overview of 45
    - process control 162
    - scheduling 147
    - scheduling request to MHP 156
    - scheduling request to SPP 147

- starting 126
- starting application from 140
- testing and debugging facilities 55
- that handles offline work 55
- transactions 67
- UAP shared library 205
- UAP trace 333
- UAP trace information 333
- uCosminexus TP1 Connector 76
- unload check 257
  - restraining 257
- unload check restraint 257
- unload journal file 257
- unload-journals file 257
- unloading
  - archive journal file 271, 370
  - journal 318
  - system journal file 257
  - TAM file 298
- unprocessed message, monitoring 141
- unprocessed receive messages, lifetime for 142
- unprocessed send message, lifetime for 141
- unrecoverable DAM file, access to 282
- unrestricted to database 7
- update, deferred 279
- uptime statistics 222, 371
- user application program 45
- user exit routine for determining application name 126
- user journal 256
  - contents of 256
- user server 20
  - assigning scheduling priority 165
  - process 162
  - providing OpenTP1 services 20
- user service definition 121

## V

- valid guarantee generation 262
- version number conventions xiv

## W

- WAN 10

## X

- X/Open DTP model 16, 307
  - transaction processing in 16
- XA interface 79
  - supported by DBMS 307
- XA resource service, transaction control based on 70
- XAR performance verification trace 335
- XAR performance verification trace information file 335
- XATMI communication service 189
- XATMI interface
  - communication mode 114
  - communication via RPC that uses 113
- XDM/DCCM3, communication with 188
- XID 71



---

# 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:

<b>Manual name:</b>	
<b>Manual number:</b>	
<b>Your name:</b>	
<b>Company or organization:</b>	
<b>Street address:</b>	
<b>Comment:</b>	

<b>(For Hitachi use)</b>
--------------------------