HITACHI
Inspire the Next

OpenTP1 Version 7

Tester and UAP Trace User's Guide

3000-3-D57-20(E)

■ Relevant program products

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

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

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.

Portions of this document are extracted from *X/Open CAE Specification System Interfaces and Headers, Issue4*, (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.

No further reproduction of this material is permitted without the prior permission of the copyright owners.

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-D57(E)): June 2006

Edition 3 (3000-3-D57-20(E)): October 2010

■ Copyright

Summary of amendments

The following table lists changes in this manual (3000-3-D57-20(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
UAP traces (UAP trace data files) can now be collected for processes even if the process is not aborted. Along with this change, the `-f` option has been added to the `uatdump` command.	1.1, 1.3, 15.1, 15.1.1, 15.2.2(1), 15.2.2(2), 15.2.3

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

The following table lists changes in the manual (3000-3-D57-10(E)) and product changes related to that manual.

Change
An explanation of specifying an environment variable in a path name for the `rpc_trace_name` definition operand has been added.

Preface

This manual describes how to use the testers and UAP trace facility of the Distributed Transaction Processing Facility OpenTP1.

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

Intended readers

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

This manual consists of five parts and an appendix, as outlined below.

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

Organization of this manual

This manual is organized into the following parts and chapters:

PART 1. Overview of Testers and UAP Traces

1. Overview

This chapter describes the types of testers and introduces UAP traces.

PART 2. Online Tester (TP1/Online Tester)

2. Facilities

This chapter describes the facilities of the online tester TP1/Online Tester for TP1/Server Base.

3. Setting the Test Environment

This chapter describes the definitions for setting the test environment to execute TP1/Online Tester.

4. Test Execution

This chapter describes how to create a test UAP, request services, and edit test information.

5. Operating Commands

This chapter describes the test operating commands.

6. Error Recovery

This chapter describes TP1/Online Tester errors and how to handle them.

PART 3. Online Tester (TP1/Message Control/Tester)

7. Facilities

This chapter describes the facilities of the online tester TP1/Message Control/ Tester for TP1/Message Control.

8. Test Execution

This chapter describes how to start and end a test, how duplicate test mode specifications are handled, and how to inherit test mode information and edit test information.

9. Operating Commands

This chapter describes the test operating commands.

PART 4. Offline Tester

10. Facilities

This chapter describes the facilities of the offline tester TP1/Offline Tester.

11. Setting the Test Environment

This chapter describes the definitions for setting the test environment to execute TP1/Offline Tester, the files that the user creates, and the files that TP1/Offline Tester creates.

12. Test Execution

This chapter describes how to create a test UAP, start and end a test, activate and terminate UAPs, request services, and edit trace information collected by the offline tester.

13. Operating Commands

This chapter describes the test operating commands and subcommands.

14. Simulation Functions

This chapter lists the processing and return values of the functions for simulating OpenTP1 functions.

PART 5. UAP Traces

15. How to Use UAP Traces

This chapter describes how to use UAP traces.

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))[#]
- *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))[#]
- *OpenTP1 Version 7 TP1/Message Queue Messages* (3000-3-D91(E))[#]
- *OpenTP1 Version 7 TP1/Message Queue Application Programming Guide* (3000-3-D92(E))[#]
- *OpenTP1 Version 7 TP1/Message Queue Application Programming Reference* (3000-3-D93(E))[#]

Other OpenTP1 products

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

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^{TM}
JP1	JP1/AJS2	JP1/AJS2 - Agent	JP1/Automatic Job Management System 2 - Agent
		JP1/AJS2 - Manager	JP1/Automatic Job Management System 2 - Manager

Abbreviation			Full name or meaning
		JP1/AJS2 - View	JP1/Automatic Job Management System 2 - View
	JP1/AJS2 - Scenario Operation	JP1/AJS2 - Scenario Operation Manager	JP1/Automatic Job Management System 2 - Scenario Operation Manager
		JP1/AJS2 - Scenario Operation View	JP1/Automatic Job Management System 2 - Scenario Operation View
		JP1/NETM/Audit	JP1/NETM/Audit - Manager
Linux			Linux(R)
Linux (AMD64/Intel EM64T/x86)			Red Hat Enterprise Linux AS 4 (AMD64 & Intel EM64T)
			Red Hat Enterprise Linux AS 4 (x86)
			Red Hat Enterprise Linux ES 4 (AMD64 & Intel EM64T)
			Red Hat Enterprise Linux ES 4 (x86)
			Red Hat Enterprise Linux 5 (AMD/Intel 64)
			Red Hat Enterprise Linux 5 (x86)
			Red Hat Enterprise Linux 5 Advanced Platform (AMD/Intel 64)
			Red Hat Enterprise Linux 5 Advanced Platform (x86)
Linux (IPF)			Red Hat Enterprise Linux AS 4 (IPF)
			Red Hat Enterprise Linux 5 (Intel Itanium)
			Red Hat Enterprise Linux 5 Advanced Platform (Intel Itanium)
MS-DOS			Microsoft(R) MS-DOS(R)
NETM/DM			JP1/NETM/DM Client
			JP1/NETM/DM Manager
			JP1/NETM/DM SubManager
Oracle			Oracle 10g
			Oracle9i
Solaris			Solaris 8
			Solaris 9

Abbreviation		Full name or meaning
		Solaris 10
TP1/Client	TP1/Client/J	uCosminexus TP1/Client/J
	TP1/Client/P	uCosminexus TP1/Client/P
	TP1/Client/W	uCosminexus TP1/Client/W
		uCosminexus TP1/Client/W(64)
TP1/EE		uCosminexus TP1/Server Base Enterprise Option
		uCosminexus TP1/Server Base Enterprise Option(64)
TP1/Extension 1		uCosminexus TP1/Extension 1
		uCosminexus TP1/Extension 1(64)
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

Abbreviation		Full name or meaning
		uCosminexus TP1/NET/High Availability(64)
TP1/NET/HSC		uCosminexus TP1/NET/HSC
TP1/NET/Library		uCosminexus TP1/NET/Library
		uCosminexus TP1/NET/Library(64)
TP1/NET/NCSB		uCosminexus TP1/NET/NCSB
TP1/NET/OSAS-NIF		uCosminexus TP1/NET/OSAS-NIF
TP1/NET/OSI-TP		uCosminexus TP1/NET/OSI-TP
TP1/NET/SLU - TypeP2	TP1/NET/ Secondary Logical Unit - TypeP2	uCosminexus TP1/NET/Secondary Logical Unit - TypeP2
TP1/NET/TCP/IP		uCosminexus TP1/NET/TCP/IP
		uCosminexus TP1/NET/TCP/IP(64)
TP1/NET/UDP		uCosminexus TP1/NET/User Datagram Protocol
TP1/NET/User Agent		uCosminexus TP1/NET/User Agent
TP1/NET/X25		uCosminexus TP1/NET/X25
TP1/NET/X25-Extended		uCosminexus TP1/NET/X25-Extended
TP1/NET/XMAP3		uCosminexus TP1/NET/XMAP3
TP1/Offline Tester		uCosminexus TP1/Offline Tester
TP1/Online Tester		uCosminexus TP1/Online Tester
TP1/Resource Manager Monitor		uCosminexus TP1/Resource Manager Monitor
TP1/Server Base		uCosminexus TP1/Server Base
		uCosminexus TP1/Server Base(64)
TP1/Shared Table Access		uCosminexus TP1/Shared Table Access
TP1/Web		uCosminexus TP1/Web
Windows 2000		Microsoft(R) Windows(R) 2000 Advanced Server Operating System
		Microsoft(R) Windows(R) 2000 Datacenter Server Operating System

Abbreviation	Full name or meaning
	Microsoft(R) Windows(R) 2000 Professional Operating System
	Microsoft(R) Windows(R) 2000 Server Operating System
Windows Server 2003	Microsoft(R) Windows Server(R) 2003, Datacenter Edition
	Microsoft(R) Windows Server(R) 2003, Enterprise Edition
	Microsoft(R) Windows Server(R) 2003, Standard Edition
Windows Server 2003 R2	Microsoft(R) Windows Server(R) 2003 R2, Enterprise Edition
	Microsoft(R) Windows Server(R) 2003 R2, Standard Edition
Windows Server 2003 x64 Editions	Microsoft(R) Windows Server(R) 2003, Datacenter x64 Edition
	Microsoft(R) Windows Server(R) 2003, Enterprise x64 Edition
	Microsoft(R) Windows Server(R) 2003, Standard x64 Edition
Windows Server 2003 R2 x64 Editions	Microsoft(R) Windows Server(R) 2003 R2, Enterprise x64 Edition
	Microsoft(R) Windows Server(R) 2003 R2, Standard x64 Edition
Windows Server 2008	Microsoft(R) Windows Server(R) 2008 Datacenter (x86)
	Microsoft(R) Windows Server(R) 2008 Enterprise (x86)
	Microsoft(R) Windows Server(R) 2008 Standard (x86)
Windows Server 2008 x64 Editions	Microsoft(R) Windows Server(R) 2008 Datacenter (x64)
	Microsoft(R) Windows Server(R) 2008 Enterprise (x64)
	Microsoft(R) Windows Server(R) 2008 Standard (x64)
Windows Vista	Microsoft(R) Windows Vista(R) Business (x86)
	Microsoft(R) Windows Vista(R) Enterprise (x86)
	Microsoft(R) Windows Vista(R) Ultimate (x86)
Windows Vista x64 Editions	Microsoft(R) Windows Vista(R) Business (x64)

Abbreviation	Full name or meaning
	Microsoft(R) Windows Vista(R) Enterprise (x64)
	Microsoft(R) Windows Vista(R) Ultimate (x64)
Windows XP	Microsoft(R) Windows(R) 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

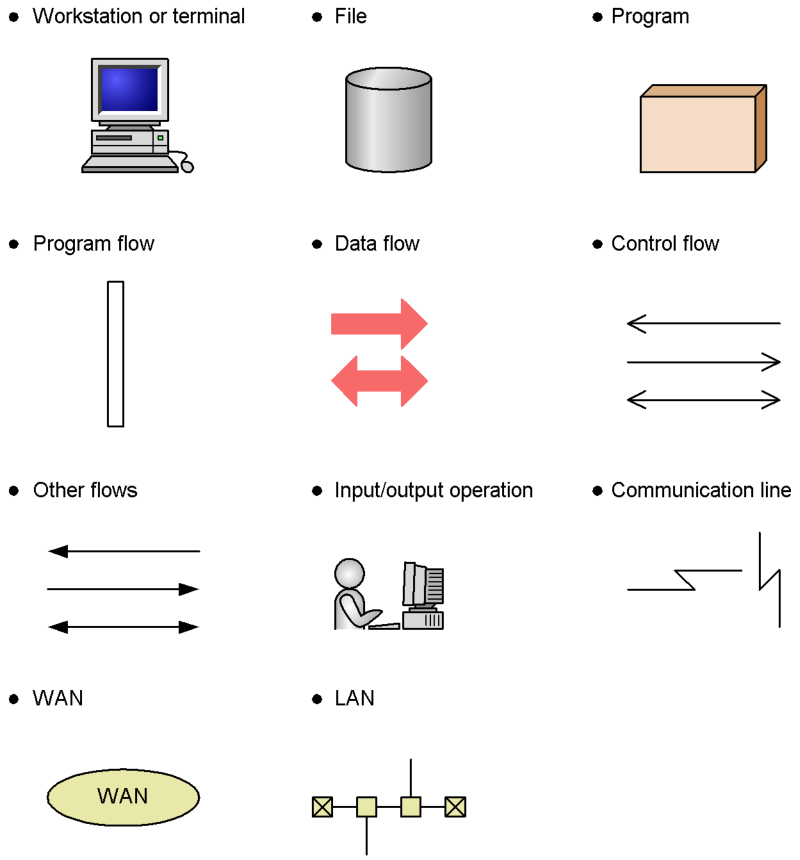
This manual also uses the following acronyms:

Acronym	Full name or meaning
ACL	Access Control List
ANSI	American National Standards Institute
AP	Application Program
API	Application Programming Interface
C/S	Client/Server
CRM	Communication Resource Manager
CUP	Client User Program
DAM	Direct Access Method
DBMS	Database Management System
DML	Data Manipulation Language
DNS	Domain Name System
FEP	Front End Processor
GUI	Graphical User Interface
HA	High Availability
HI-ODTP	Hitachi - Open Distributed Transaction Processing Adapter
ISAM	Indexed Sequential Access Method
IST	Internode Shared Table

Acronym	Full name or meaning
LAN	Local Area Network
MCF	Message Control Facility
MHP	Message Handling Program
MQA	Message Queue Access
MQI	Message Queue Interface
NIF/HNA	Network Interface Feature/Hitachi Network Architecture
NIF/OSI	Network Interface Feature/OSI
OS	Operating System
OSI	Open Systems Interconnection
OSI TP	Open Systems Interconnection Transaction Processing
PC	Personal Computer
PRF	Performance
RM	Resource Manager
RPC	Remote Procedure Call
SPP	Service Providing Program
STDL	Structured Transaction Definition Language
SUP	Service Using Program
TAM	Table Access Method
TCP/IP	Transmission Control Protocol/Internet Protocol
TM	Transaction Manager
UAP	User Application Program
UOC	User Own Coding
WAN	Wide Area Network
WS	Workstation

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
`/BeTRAN`	`/BeTRAN`	`/opt/OpenTP1`	The directory in which OpenTP1 was installed

Conventions: Fonts and symbols

The following table explains the fonts used in this manual:

Font	Convention
Bold	**Bold** type indicates text on a window, other than the window title. Such text includes menus, menu options, buttons, radio box options, or explanatory labels. For example: • From the **File** menu, choose **Open**. • Click the **Cancel** button. • In the **Enter name** entry box, type your name.
Italics	*Italics* are used to indicate a placeholder for some actual text to be provided by the user or system. For example: • Write the command as follows: `copy` *source-file target-file* • The following message appears: `A file was not found. (file =` *file-name*`)` *Italics* are also used for emphasis. For example: • Do *not* delete the configuration file.
`Code font`	A `code font` indicates text that the user enters without change, or text (such as messages) output by the system. For example: • At the prompt, enter `dir`. • Use the `send` command to send mail. • The following message is displayed: `The password is incorrect.`
`SD`	Bold code-font characters indicate the abbreviation for a command.
`perm`	Underlined characters indicate the default value.

The following table explains the symbols used in this manual:

<table>
<tr><th>Symbol</th><th>Convention</th></tr>
<tr><td>|</td><td>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.</td></tr>
<tr><td>{ }</td><td>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.</td></tr>
<tr><td>[ ]</td><td>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.</td></tr>
</table>

Symbol	Convention
`...`	In coding, an ellipsis (...) indicates that one or more lines of coding are not shown for purposes of brevity. In syntax explanations, an ellipsis indicates that the immediately preceding item can be repeated as many times as necessary. For example: `A, B, B, ...` means that, after you specify `A, B,` you can specify B as many times as necessary.
~	The item preceding this symbol must be specified according to the rule given in the angle brackets (< >) following this symbol.
<< >>	Default value assumed when a specification is omitted.
< >	Information between these symbols indicates the syntax of the item.
(( ))	Range of specifiable values.

Conventions for permitted characters

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

Type	Definition
Upper-case alphabetic characters	`A` to `Z`
Lower-case alphabetic characters	`a` to `z`
Alphabetic characters	`A` to `Z`, `a` to `z`
Numeric characters	`0` to `9`
Alphanumeric characters	`A` to `Z`, `a` to `z`, `0` to `9`
Unsigned integer	Numeric values `0` to `9`
Hexadecimal	Numeric values `0` to `9`, `A` to `F`, and `a` to `f`
Identifier	String of alphanumeric characters, beginning with an alphabetic character `A` to `Z` or `a` to `z`
Symbolic name	String of alphanumeric symbols, beginning with an alphabetic symbol
Pathname	Symbolic names, slashes (`/`), and periods (`.`), depending on the operating system

Conventions: KB, MB, GB, and TB

This manual uses the following conventions:

- 1 KB (kilobyte) is 1,024 bytes.

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

Conventions: Platform-specific notational differences

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

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

Conventions: Version numbers

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

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

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

Important note on this manual

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

Contents

Preface i

Intended readers i
Organization of this manual i
Related publications iii
Conventions: Abbreviations for product names iv
Conventions: Acronyms ix
Conventions: Diagrams x
Conventions: Differences in installation directory paths xi
Conventions: Fonts and symbols....................xi
Conventions: KB, MB, GB, and TB xiii
Conventions: Platform-specific notational differences xiv
Conventions: Version numbers....................xiv
Important note on this manual....................xiv

PART 1: Overview of Testers and UAP Traces

1. Overview 1

1.1 Testers and UAP traces....................2
1.2 Overview of testers....................3
1.2.1 Online testers....................3
1.2.2 Offline tester (TP1/Offline Tester) 6
1.3 Overview of UAP traces....................9

PART 2: Online Tester (TP1/Online Tester)

2. Facilities 11

2.1 Facilities of the online tester....................12
2.2 Simulating a client UAP....................13
2.2.1 Simulating a client UAP with an RPC interface 13
2.2.2 Simulating a client UAP with an XATMI interface 14
2.3 Simulating a server UAP 18
2.3.1 Simulating a server UAP with an RPC interface 18
2.3.2 Simulating a server UAP with an XATMI interface 19
2.4 Simulating the MCF 22
2.4.1 MCF simulation functions....................22
2.4.2 Simulating message send/receive....................22
2.4.3 Simulating continuous inquiry responses....................23

2.4.4 Simulating application program startup requests 25
2.4.5 Simulating synchronous point processing 27
2.5 Disabling resource updating 28
2.6 Simulating operating commands 29
2.7 Creating and outputting tester files 31
2.7.1 Creating tester files 31
2.7.2 Editing and outputting tester files 33
2.8 Collecting test information 34
2.8.1 Collecting UAP trace information 34
2.8.2 Merging, editing, and outputting UAP trace information 35
2.8.3 Editing send messages 39
2.9 Interlocking the debugger 40

3. Setting the Test Environment 43

3.1 System definitions for the online tester 44
3.1.1 System service configuration definition 44
3.1.2 Tester service definition 44
3.1.3 Tester service definition (command format) 47
3.1.4 User service definition 48
3.1.5 Setting the typed buffer 54
3.1.6 Setting send/receive procedures 56
3.2 Setting environment variables 59
3.3 User-created files 60
3.3.1 Service request data files 62
3.3.2 Service response data files 65
3.3.3 XATMI receive data file 68
3.3.4 MCF receive message files 71
3.3.5 Operating command result data file 81
3.4 Creating files 84
3.4.1 Test directory 84
3.4.2 Test data definition file 84
3.4.3 Files created by the online tester 95

4. Test Execution 99

4.1 Creating UAPs 100
4.2 Service requests to an SPP 102
4.2.1 Client UAP simulator 102
4.2.2 Server UAP simulator 102
4.3 Service requests to an MHP 103
4.4 Creating tester files 104
4.4.1 Creating tester files using the test data definition file 104
4.4.2 Creating tester files using operating command output data 105
4.5 Editing test information 107
4.5.1 Displaying test status 107

4.5.2 Collecting UAP trace information....107
4.5.3 Merging and outputting UAP trace information....108
4.5.4 UAP traces for MCF simulation functions....109
4.5.5 Editing and outputting send messages....109
4.5.6 Checking UAP response data....110
4.5.7 Checking UAP send data....110

5. Operating Commands 111

5.1 Operating commands for running tests....112
5.1.1 utodbgstop (termination of a UAP interlocked with the debugger)....112
5.1.2 utodebug (activation of a UAP interlocked with the debugger)....113
5.1.3 utofilcre (tester file creation)....115
5.1.4 utofilout (edited output of the tester file content)....116
5.1.5 utols (test status display)....129
5.1.6 utomhpsvc (service requests to an MHP)....130
5.1.7 utomsgout (edited output of send messages)....131
5.1.8 utosppsvc (service requests to an RPC interface SPP)....136
5.1.9 utotrcmrg (merger of UAP trace information)....137
5.1.10 utotrcout (edited output of UAP trace information)....138
5.1.11 utoxsppsvc (service requests to an XATMI interface SPP)....150

6. Error Recovery 153

6.1 Handling online tester errors....154
6.1.1 Error conditions and causes....154
6.1.2 Online tester errors....155
6.1.3 File errors....156
6.1.4 UAP errors....156

PART 3: Online Tester (TP1/Message Control/Tester)

7. Facilities 159

7.1 MHP testing....160
7.1.1 Disabling updating of non-MCF resources....160
7.1.2 Invalidating send messages....160
7.1.3 Invalidating application startup messages....161
7.1.4 Suppressing error events....161
7.1.5 Suppressing MHP automatic shutdown....162
7.2 Collecting test information....164
7.2.1 Collecting UAP trace information....164

8. Test Execution 165

8.1 Starting and ending a test....166
8.1.1 Starting a test and setting the test environment....166

8.1.2 Ending a test 167
8.2 Duplicate test mode specifications 168
8.3 Inheriting test mode information 169
8.4 Editing test information 170
8.4.1 Displaying test mode information 170
8.4.2 Collecting UAP trace information 170
8.4.3 Merging and outputting UAP trace information 170

9. Operating Commands 173

9.1 Operating commands for running tests 174
9.1.1 mcfutfst (MCF online tester use declaration) 174
9.1.2 mcflsutf (display of MCF online tester status) 174
9.2 Operating commands for testing a logical terminal 176
9.2.1 mcftulsle (display of test mode information for a logical terminal) 176
9.2.2 mcftules (start of a logical terminal test) 178
9.2.3 mcftulee (termination of a logical terminal test) 180
9.3 Operating commands for testing an application 181
9.3.1 mcfaulsap (display of test mode information for an application) 181
9.3.2 mcfauaps (start of an application test) 184
9.3.3 mcfauape (termination of an application test) 187
9.4 Operating commands for testing a service group 189
9.4.1 mcftulssg (display of test mode information for a service group) 189
9.4.2 mcftusgs (start of a service group test) 191
9.4.3 mcftusge (termination of a service group test) 193

PART 4: Offline Tester

10. Facilities 195

10.1 Facilities of the offline tester 196
10.2 Simulating a client UAP 197
10.2.1 Simulating a client UAP with an RPC interface 197
10.2.2 Simulating a client UAP with an XATMI interface 198
10.2.3 Simulating a client UAP with a TxRPC interface 198
10.3 Simulating a server UAP 199
10.3.1 Simulating a server UAP with an RPC interface 200
10.3.2 Simulating a server UAP with an XATMI interface 200
10.3.3 Simulating a server UAP with a TxRPC interface 200
10.4 Simulating the MCF 202
10.5 Simulating file services 203
10.5.1 Simulating the DAM service 203
10.5.2 Simulating the TAM service 204
10.6 Simulating OpenTP1 functions 206
10.7 Simulating operating commands 207

10.8 Creating tester files..208
10.9 Continuous command execution 209
10.10 Debugger connection..210
10.11 Collecting test information 211
10.11.1 Collecting offline tester trace information 211

11. Setting the Test Environment 213

11.1 System definitions for the offline tester...214
11.1.1 Offline tester environment definition 214
11.1.2 User service definition...231
11.1.3 Setting function return values..232
11.1.4 Setting continuous execution commands 236
11.1.5 Creating stubs 238
11.2 User-created files 239
11.2.1 Service request data files...240
11.2.2 Service response data files...244
11.2.3 XATMI receive data file...248
11.2.4 MCF receive message files...250
11.2.5 DAM file 255
11.2.6 TAM file 256
11.2.7 Operating command result data file 257
11.3 Creating files..259
11.3.1 Test data definition file..259
11.3.2 Files created by the offline tester...270

12. Test Execution 271

12.1 Creating UAPs..272
12.1.1 Creating UAP execution format programs...272
12.2 Starting and ending an offline test..277
12.3 Activating and terminating UAPs 278
12.4 Service requests 279
12.5 Creating tester files..280
12.6 Continuous command execution 281
12.7 Debugger connection...282
12.8 Editing offline tester trace information 283
12.9 Notes on running tests 284
12.9.1 Notes on the offline tester 284
12.9.2 Notes on files...288
12.9.3 Notes on UAPs..289

13. Operating Commands 291

13.1 Operating commands for running tests...292
13.1.1 utfdamcre (creation of offline tester DAM file).......................................292
13.1.2 utffilcre (tester file creation)..293

13.1.3 utfstart (offline tester startup) 293
13.1.4 utftamcre (creation of offline tester TAM files) 295
13.1.5 utftrcpic (retrieval of offline tester trace information) 296
13.2 Subcommands for running tests 302
13.2.1 call (service request) 302
13.2.2 cmdauto (continuous command execution) 303
13.2.3 end (offline tester termination) 304
13.2.4 ps (test status display) 304
13.2.5 read (input of tester file name to offline tester) 305
13.2.6 start (service group activation) 306
13.2.7 stop (service group termination) 307
13.2.8 write (input of tester file name to offline tester) 308

14. Simulation Functions 309

14.1 List of simulation functions and processing 310
14.2 List of return values for simulation functions 326
14.3 List of functions not supported by the simulation feature 342

PART 5: UAP Traces

15. How to Use UAP Traces 347

15.1 Collecting UAP traces 348
15.1.1 UAP trace collection units 348
15.1.2 Trace area definition 348
15.1.3 Information to collect 348
15.2 Editing and outputting UAP traces 349
15.2.1 UAP trace output units 349
15.2.2 UAP trace output methods 350
15.2.3 uatdump (edited output of UAP trace) 352
15.2.4 UAP trace output format 354

Index 359

List of figures

Figure 1-1: OpenTP1 testers ... 3
Figure 1-2: Overview of online tester ... 5
Figure 1-3: Overview of MCF online tester ... 6
Figure 1-4: Overview of offline tester ... 8
Figure 2-1: Simulating a client UAP with an RPC interface ... 14
Figure 2-2: Simulating a client UAP for request/response service paradigm ... 15
Figure 2-3: Simulating a client UAP for conversational service paradigm ... 17
Figure 2-4: Simulating a server UAP with an RPC interface ... 19
Figure 2-5: Simulating a server UAP for request/response service paradigm ... 20
Figure 2-6: Simulating a server UAP for conversational service paradigm ... 21
Figure 2-7: Simulating message send/receive ... 23
Figure 2-8: Simulating continuous inquiry responses ... 24
Figure 2-9: Simulating an application program startup request ... 26
Figure 2-10: Replacing command execution results ... 30
Figure 2-11: Result of merging UAP trace information ... 36
Figure 2-12: Collecting, merging, editing, and outputting UAP trace information ... 38
Figure 2-13: Interlocking the debugger ... 41
Figure 3-1: Receive data and tester files ... 70
Figure 7-1: Example of transaction processing from message receive to message send ... 162
Figure 10-1: Simulating a client UAP ... 197
Figure 10-2: Simulating a server UAP ... 200
Figure 10-3: Simulating an MCF ... 202
Figure 10-4: Simulating the DAM service ... 204
Figure 10-5: Simulating the TAM service ... 205
Figure 10-6: Simulating UAP operating commands ... 207
Figure 10-7: Continuous command execution ... 209
Figure 10-8: Debugger connection ... 210
Figure 10-9: Collecting offline tester trace information ... 212
Figure 12-1: Procedure for creating UAP execution format program with the RPC or XATMI interface ... 273
Figure 12-2: Procedure for creating UAP execution format program with the TxRPC interface ... 275
Figure 12-3: Recursive calls using the offline tester ... 287
Figure 15-1: Inter-UAP communication and collected UAP traces ... 349
Figure 15-2: Overview of automatic edit and output of UAP trace ... 351
Figure 15-3: Overview of editing and outputting UAP trace to standard output by a command ... 352

List of tables

Table 2-1: Tester files created by tester file creation facility 31
Table 2-2: Kinds of tester files to be created, available data extraction commands, and available data 32
Table 2-3: Tester files available for edit and output with the tester file edit and output facility.. 33
Table 2-4: Functions that can use the complete I/O data trace collection facility 34
Table 3-1: test_mode specifications and available test facilities... 50
Table 3-2: Relationships between calling UAP and called UAP when requesting services.... 50
Table 3-3: List of tester files to be created by the user 60
Table 3-4: Names for user-created tester files.. 61
Table 3-5: Keywords and input data formats for RPC request data files................................ 90
Table 3-6: Keywords and input data formats for XATMI request data files........................... 90
Table 3-7: Keywords and input data formats for RPC response data files 91
Table 3-8: Keywords and input data formats for XATMI response data files 91
Table 3-9: Keywords and input data formats for XATMI receive data files........................... 92
Table 3-10: Keywords and input data formats for asynchronous receive message files.......... 93
Table 3-11: Keywords and input data formats for synchronous receive message files 93
Table 3-12: Keywords and input data formats for operating command result data file........... 95
Table 3-13: List of files created by online tester 95
Table 3-14: Names for tester files created by the online tester 96
Table 4-1: Dummy values and non-collectable trace information 109
Table 5-1: List of operating commands..112
Table 6-1: Online tester errors and causes 154
Table 6-2: Time-out error events caused by a debugger-interlocked UAP and related definitions 157
Table 8-1: Duplicate test mode specifications.. 168
Table 8-2: Inheritance of test mode information.. 169
Table 9-1: List of operating commands.. 174
Table 9-2: Operating commands for running tests on a logical terminal.............................. 176
Table 9-3: Operating commands for running tests on an application 181
Table 9-4: IDs to be specified when testing ERREVT (mcfauaps command)....................... 184
Table 9-5: IDs to be specified when testing ERREVT (mcfauape command)....................... 187
Table 9-6: Operating commands for running tests on an application 189
Table 10-1: Tester files created by tester file creation facility 208
Table 11-1: Format errors and validity of definitions 215
Table 11-2: List of user-created files.. 239
Table 11-3: RPC request data file keywords and input data formats 265
Table 11-4: XATMI request data file keywords and corresponding input data formats 265
Table 11-5: TxRPC request data file keywords and corresponding input data format 266
Table 11-6: RPC response data file keywords and corresponding input data formats........... 266

Table 11-7: XATMI response data file keywords and corresponding input data formats 267
Table 11-8: TxRPC response data file keywords and corresponding input data format.........267
Table 11-9: XATMI receive data file keywords and input data formats.................................267
Table 11-10: MCF receive message file keywords and corresponding input data formats 268
Table 11-11: Operation command result data file keywords and corresponding input data formats...269
Table 11-12: List of files created by offline tester 270
Table 12-1: Upper limits of offline tester...285
Table 13-1: List of operating commands for offline testing 292
Table 13-2: List of subcommands for offline testing..302
Table 14-1: List of offline tester simulation functions..310
Table 14-2: List of return values for simulation functions 326
Table 14-3: List of functions not supported by the simulation feature (for C).......................342
Table 14-4: List of functions not supported by the simulation feature (for COBOL)............343
Table 15-1: Directories and file names of core file and UAP trace output file 350

Chapter

1. Overview

This chapter introduces the testers and UAP traces provided by OpenTP1.

This chapter contains the following sections:

1.1 Testers and UAP traces
1.2 Overview of testers
1.3 Overview of UAP traces

1.1 Testers and UAP traces

OpenTP1 provides *test support programs (testers)* for checking UAP operation. OpenTP1 also provides a troubleshooting facility, the *UAP trace facility*, for troubleshooting UAP operation.

The OpenTP1 testers include online testers which operate in an online environment with TP1/Server Base or TP1/Message Control and an offline tester used in an offline environment.

The UAP trace facility can be used with TP1/Server Base.

Each tester requires a different program product, as follows:

TP1/Online Tester

For using the TP1/Server Base online tester

TP1/Message Control/Tester

For using the TP1/Message Control online tester

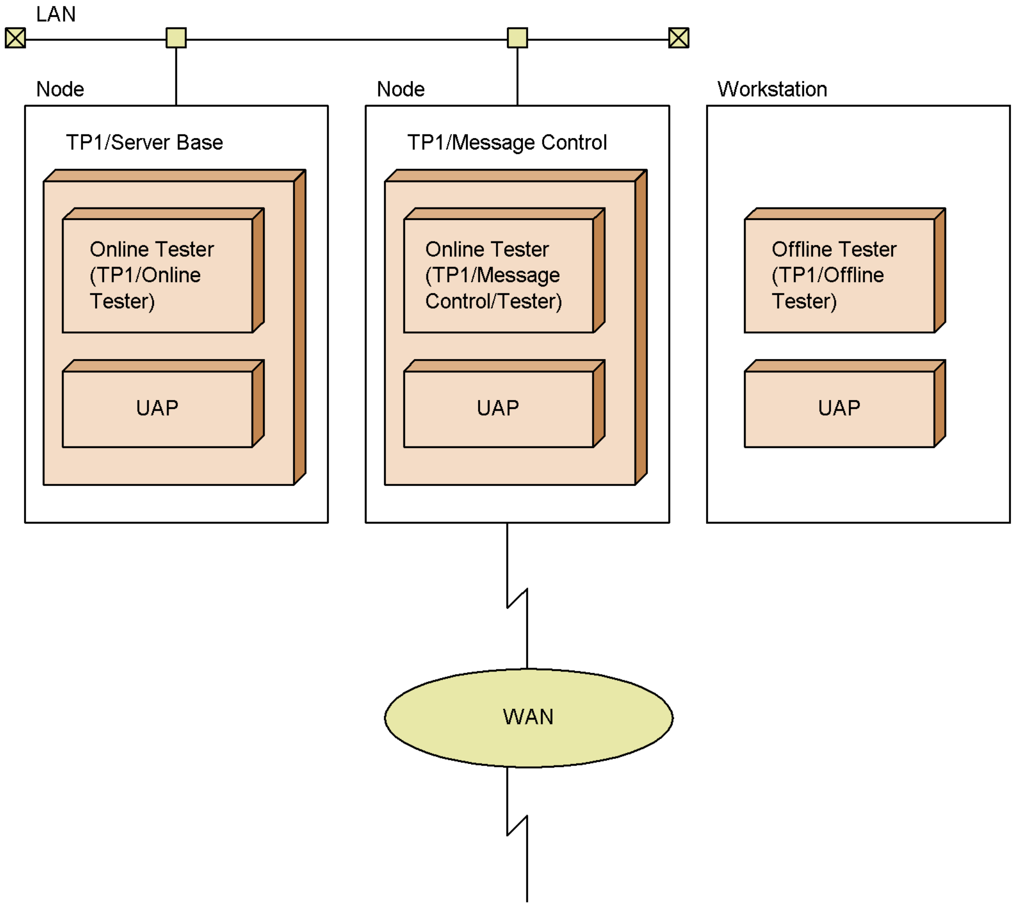
TP1/Offline Tester

For using the offline tester

1.2 Overview of testers

Figure 1-1 shows the OpenTP1 testers.

Figure 1-1: OpenTP1 testers



1.2.1 Online testers

(1) Online tester (TP1/Online Tester)

The online tester for TP1/Server Base (hereafter called the *online tester*) performs the following (see Part II for details):

- Simulates client and server UAPs
- Simulates the MCF

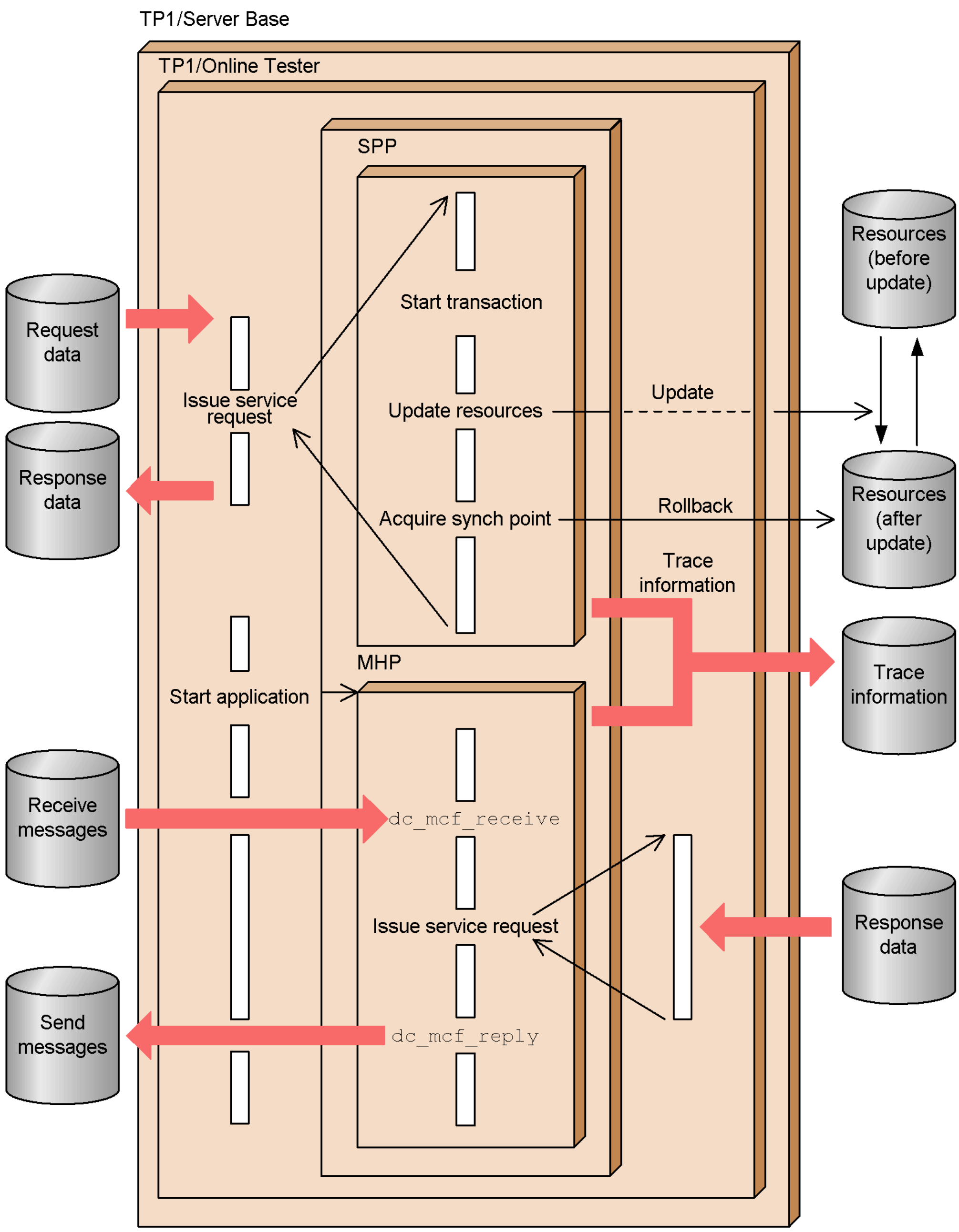
- Disables resource updating
- Simulates operating commands issued from the UAP
- Creates, edits, and outputs tester files (data files used in tests)
- Collects, edits, and outputs UAP trace information
- Collects and edits UAP send messages
- Runs with the debugger

Using the online tester, you can test and check the operation of an SUP, SPP, or MHP in an online environment.

TP1/Server Base must be installed to use the online tester.

Figure 1-2 shows how the online tester is structured.

Figure 1-2: Overview of online tester



(2) Online tester (TP1/Message Control/Tester)

The online tester for TP1/Message Control (hereafter called the *MCF online tester*) performs the following (see Part III for details):

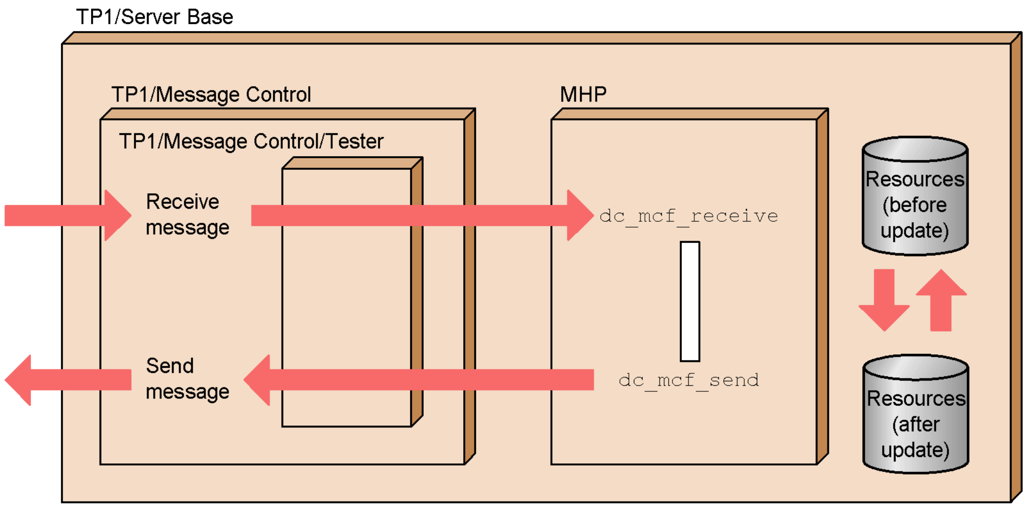
- Disables updating of non-MCF resources
- Invalidates send messages
- Invalidates application startup messages
- Suppresses error events
- Suppresses MHP automatic shutdown
- Collects UAP trace information

The online tester (TP1/Online Tester) is required for collecting UAP trace information. Otherwise, the MCF online tester can be used without installing TP1/Online Tester.

When an MHP is specified as a test program for both the online tester and the MCF online tester, the MCF online tester specification takes precedence.

Figure 1-3 shows how the MCF online tester is structured.

Figure 1-3: Overview of MCF online tester



1.2.2 Offline tester (TP1/Offline Tester)

The offline tester performs the following (see Part IV for details):

- Simulates client and server UAPs
- Simulates the MCF

- Simulates file services
- Simulates operating commands issued from the UAP
- Creates tester files (data files used in tests)
- Executes commands continuously
- Runs with the debugger
- Collects offline tester trace information

Using the offline tester, you can test and check the operation of an SPP or MHP in an offline environment.

Depending on the functions used during testing, the UAP may need to be compiled using the header files provided by the following program products:

- TP1/Server Base

  When using functions provided by TP1/Server Base
- TP1/Message Control

  When using message send/receive functions
- TP1/FS/Direct Access

  When using DAM service functions
- TP1/FS/Table Access

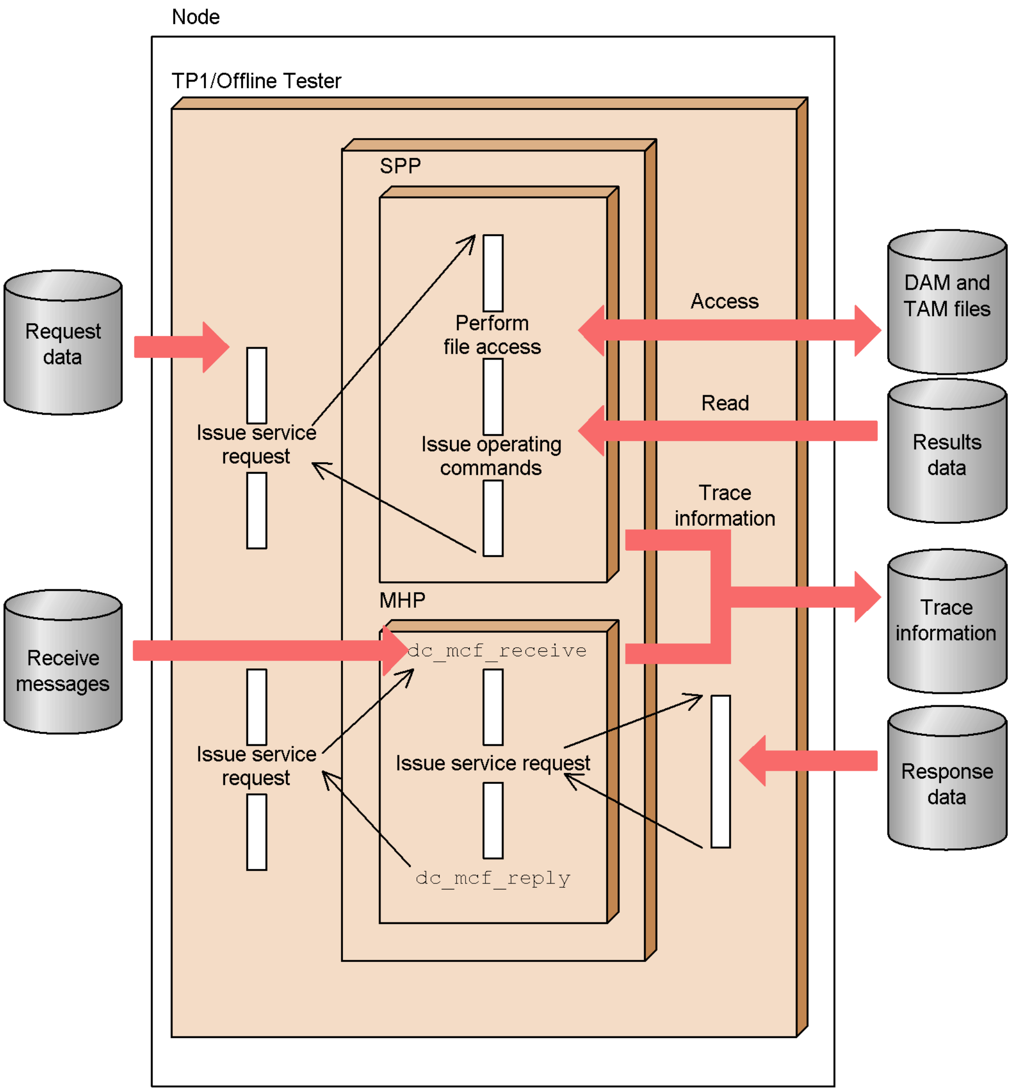
  When using TAM service functions
- TP1/Shared Table Access

  When using IST service functions

Also, the OpenTP1 `stbmake` command is required when creating a UAP for offline tester use. At UAP creation, copy the OpenTP1 command file containing the `stbmake` command.

Figure 1-4 shows how the offline tester is structured.

Figure 1-4: Overview of offline tester



1.3 Overview of UAP traces

As an aid to handling possible UAP errors, OpenTP1 collects a log of the library functions used by the UAP. This information shows which functions returned an error and which resources the UAP attempted to access. By editing and outputting this information, the user can analyze the cause of UAP errors and then correct the UAP or rebuild the system. This facility is called the *UAP trace facility*.

UAP traces are collected for each SUP, SPP, or MHP process.

If either of the following files is available when a UAP terminates abnormally, the UAP traces are automatically edited and output to that file.

- UAP trace data file
- Core file

If a UAP terminates abnormally and a core file exists, the UAP trace is automatically edited and output to a file. The user can edit and output the UAP trace to the standard output by using the `uatdump` command of TP1/Server Base. See Subsection *15.2.2 UAP trace output methods* for details of the `uatdump` command.

UAP traces can also be collected when using an online or offline tester to test a UAP. Such information is useful for analyzing the processing flow in a UAP test.

For the online tester, UAP traces are collected for TP1/Server Base. For the offline tester, specialized trace information is collected.

Chapter
2. Facilities

This chapter describes the test facilities available with the online tester.

This chapter contains the following sections:

2.1 Facilities of the online tester
2.2 Simulating a client UAP
2.3 Simulating a server UAP
2.4 Simulating the MCF
2.5 Disabling resource updating
2.6 Simulating operating commands
2.7 Creating and outputting tester files
2.8 Collecting test information
2.9 Interlocking the debugger

2.1 Facilities of the online tester

The online tester provides the following facilities for testing UAPs:

1. Client UAP simulator

   Simulates client UAP processing so that a server UAP can be tested without a client UAP.

2. Server UAP simulator

   Simulates server UAP processing so that a client UAP can be tested without a server UAP.

3. MCF simulator

   Simulates message send and receive processing controlled by TP1/Message Control so that an MHP or an SPP called by service requests from the MHP can be tested without TP1/Message Control.

4. Disabling resource update

   Disables update processing of resources so that the test UAP does not update resources used by applications.

5. Operating command simulator

   Simulates the processing of operating commands issued by a test UAP.

6. Tester file creation and editing

   Creates tester files needed for each simulation and outputs them in an edited format.

7. UAP trace collection

   Collects UAP trace information for the UAP being tested.

8. Merger and editing of UAP trace information

   Merges UAP trace information collected in multiple files and edits the information for output.

9. Send message editing

   Collects send messages from test UAPs and edits the messages for output.

10. Debugger interlocking

    Executes a UAP to be tested under control of the debugger.

2.2 Simulating a client UAP

The online tester can take the place of a client UAP in requesting services from a server UAP. This allows the user to test the server UAP without needing a client UAP. This facility is called the *client UAP simulator*.

An online tester command is used to simulate a client UAP. Before executing the command, the user must first create the processing data to be passed to the server UAP. This data is created in a *service request data file*. The response data from the server UAP is saved to the *service response data file* specified in the command.

There are two types of service request data files which are used according to the client interface:

- RPC request data file (for simulating a UAP that has an RPC interface)
- XATMI request data file (for simulating a UAP that has an XATMI interface)

There are also two types of service response data files, selected according to the type of simulated client UAP:

- RPC response data file (for simulating a UAP that has an RPC interface)
- XATMI response data file (for simulating a UAP that has an XATMI interface)

To test a server UAP using the client UAP simulator, the user must first define the server UAP as a *test-only UAP* in a user service definition. A test-only UAP is a UAP that runs in *test mode*. All of the facilities of the online tester are available for a test-only UAP.

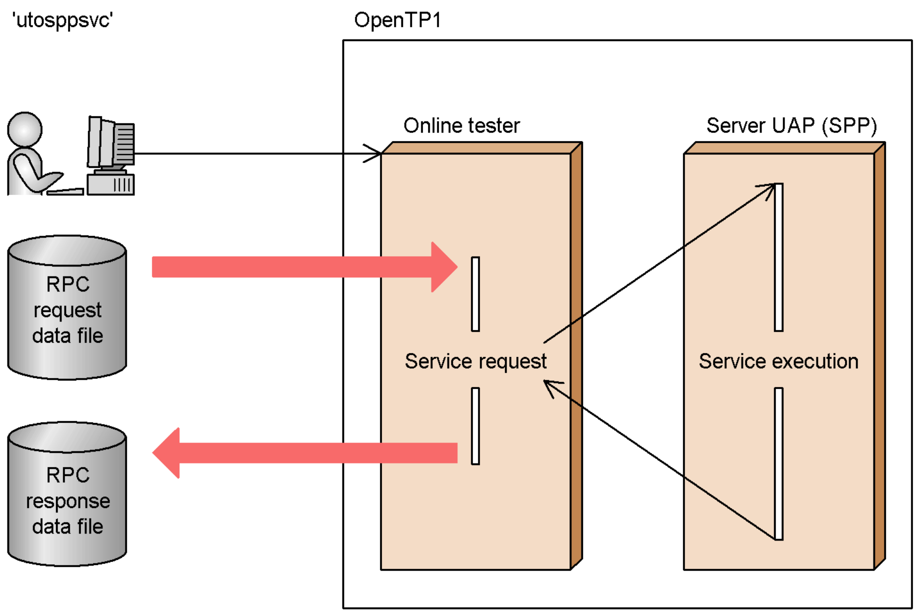
Instead of defining the server UAP as a test-only UAP, the server UAP can be defined as a usable UAP in the user service definition. A usable UAP is a SPP that runs in test mode only when the UAP being tested makes a service request.

2.2.1 Simulating a client UAP with an RPC interface

To simulate a client UAP that uses an RPC interface to send service requests, the user must first create an *RPC request data file* with the processing data to be passed to the server UAP. The response data from the server UAP is saved to the *RPC response data file* specified in the online tester command.

Figure 2-1 illustrates the client UAP simulator for an RPC interface.

Figure 2-1: Simulating a client UAP with an RPC interface



2.2.2 Simulating a client UAP with an XATMI interface

The client UAP simulator is also available when using the online tester for service requests (the request/response service paradigm and the conversational service paradigm) in an XATMI interface.

(1) Request/response service paradigm

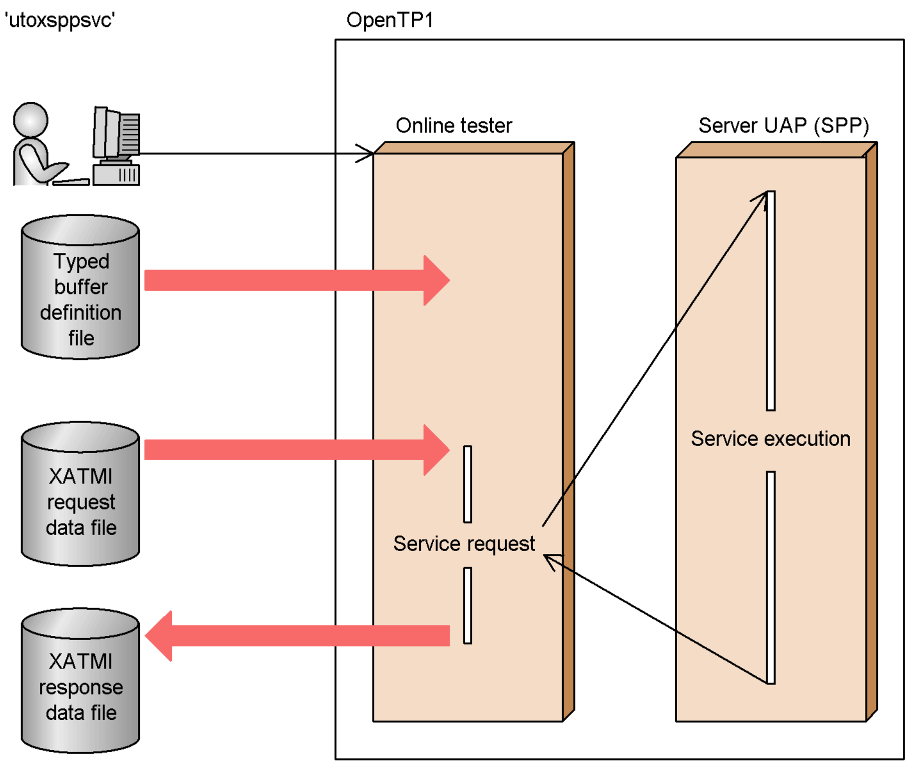
To simulate a client UAP that sends the request/response service paradigm, the user must first create an *XATMI request data file* with the processing data to be passed to the server UAP. The response data from the server UAP is saved to the *XATMI response data file* specified in the online tester command.

The user must also set the typed buffer information, needed for using the XATMI, in the *typed buffer definition file*.

Also, the types of functions to be used in the request/response service paradigm must be set as headers in the XATMI request data file.

Figure 2-2 illustrates the client UAP simulator for the request/response service paradigm.

Figure 2-2: Simulating a client UAP for request/response service paradigm



(2) Conversational service paradigm

To simulate a client UAP that sends the conversational service paradigm, the user must first create an XATMI request data file with the processing data to be passed to the server UAP. The types of functions to be used in the conversational service paradigm must be set as the file headers. The response data from the server UAP is saved to the XATMI response data file specified in the online tester command.

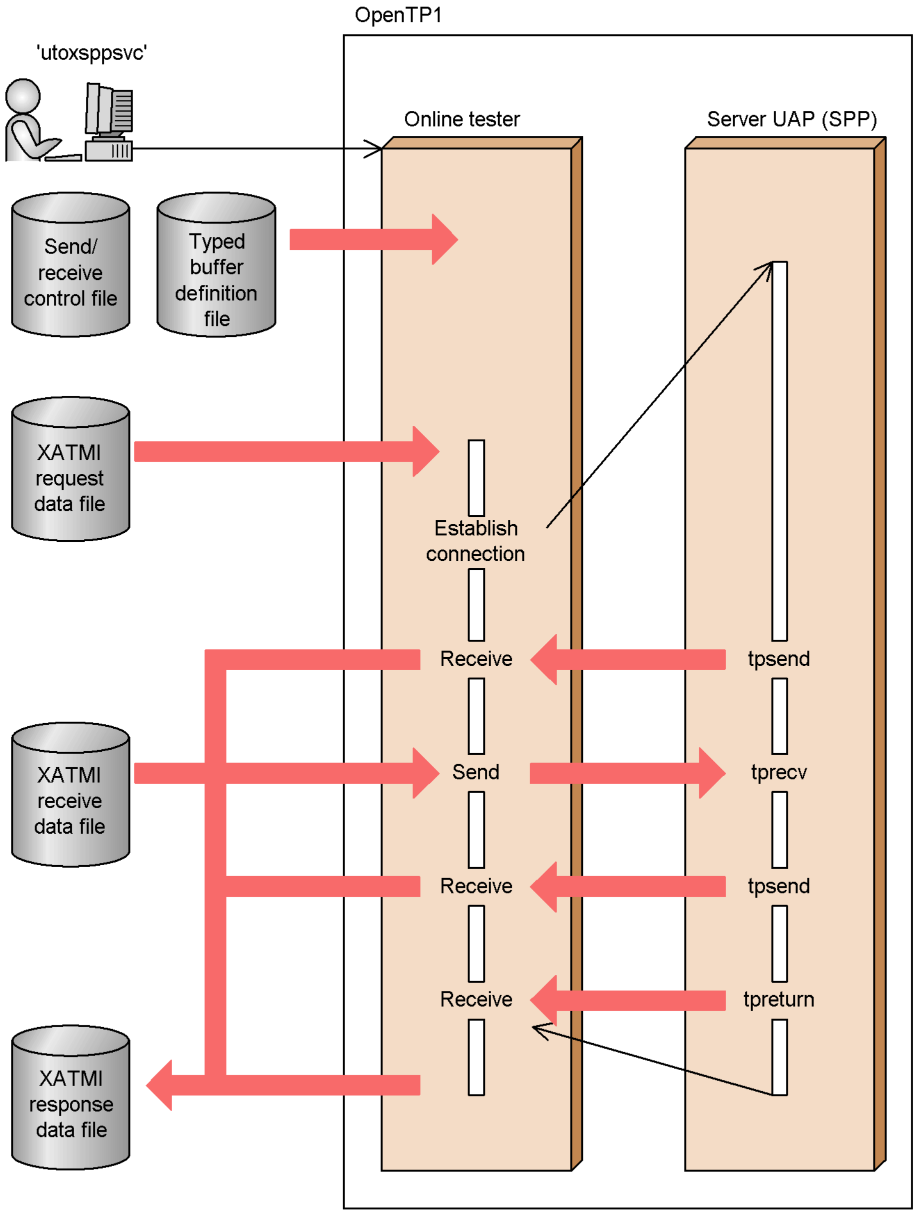
The user must also set the typed buffer information, needed for accessing the XATMI, in the typed buffer definition file.

Also, the send/receive procedures must be set in a *send/receive control file*. The user creates an *XATMI receive data file* with the data received by the test server UAP when a service is requested. The name of this file is specified in the send/receive control file. Data sent by the server UAP is saved to the XATMI response data file in the same way as response data.

The server UAP's response data and send data, which the client UAP simulator saved to the XATMI response data file, can be used by the server UAP simulator as the request data and receive data sent to a client UAP. To enable the server UAP simulator to access the response data and send data, first use the binary editor to recreate the XATMI response data file as an XATMI request data file and XATMI receive data file.

Figure 2-3 illustrates the client UAP simulator for the conversational service paradigm.

Figure 2-3: Simulating a client UAP for conversational service paradigm



2.3 Simulating a server UAP

The online tester can take the place of a server UAP in executing services requested by a client UAP. This allows the user to test the client UAP without needing a server UAP. This facility is called the *server UAP simulator*.

To simulate a server UAP, the user activates the server UAP (dummy) and then executes an OpenTP1 command. Before executing the command, the user must create the response data to be passed to the client UAP. This data is created in a *service response data file*. When the client UAP sends a service request, the online tester reads the response data from the file and passes it to the client UAP.

There are two types of service response data files which are used according to the UAP interface:

- RPC response data file (for simulating a UAP that has an RPC interface)
- XATMI response data file (for simulating a UAP that has an XATMI interface)

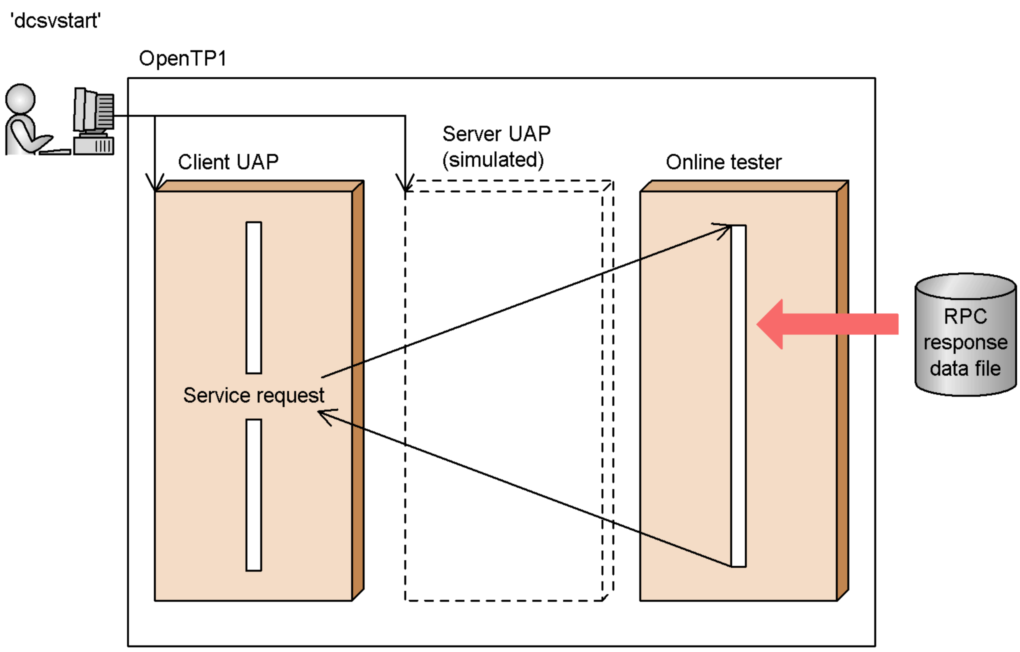
To test a client UAP using the server UAP simulator, the user must first define the server UAP as a *dummy SPP* in a user service definition. A dummy SPP is an SPP that does not actually generate processes when activated by the server UAP simulator. The dummy SPP must be activated before entering the command to start testing.

2.3.1 Simulating a server UAP with an RPC interface

To simulate a server UAP that uses an RPC interface for accepting service requests, the user must first create an *RPC response data file* with the response data to be returned to the client UAP. When the client UAP sends a service request, the online tester reads the response data from the file and returns it to the client UAP.

Figure 2-4 illustrates the server UAP simulator for an RPC interface.

Figure 2-4: Simulating a server UAP with an RPC interface



2.3.2 Simulating a server UAP with an XATMI interface

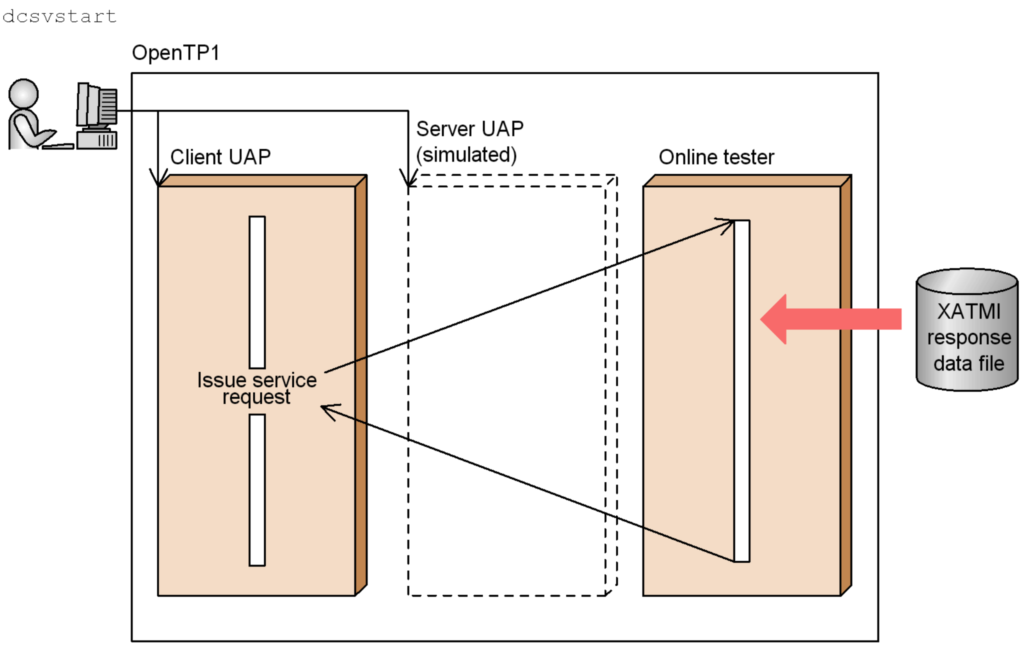
The server UAP simulator is also available when using the online tester for service requests (request/response service paradigm and conversational service paradigm) in an XATMI interface.

(1) Request/response service paradigm

To simulate a server UAP that accepts request/response service paradigm, the user must first create an *XATMI response data file* with the response data to be returned to the client UAP. When the client UAP sends a service request, the online tester reads the response data from the file and returns it to the client UAP.

Figure 2-5 illustrates the server UAP simulator for the request/response service paradigm

Figure 2-5: Simulating a server UAP for request/response service paradigm



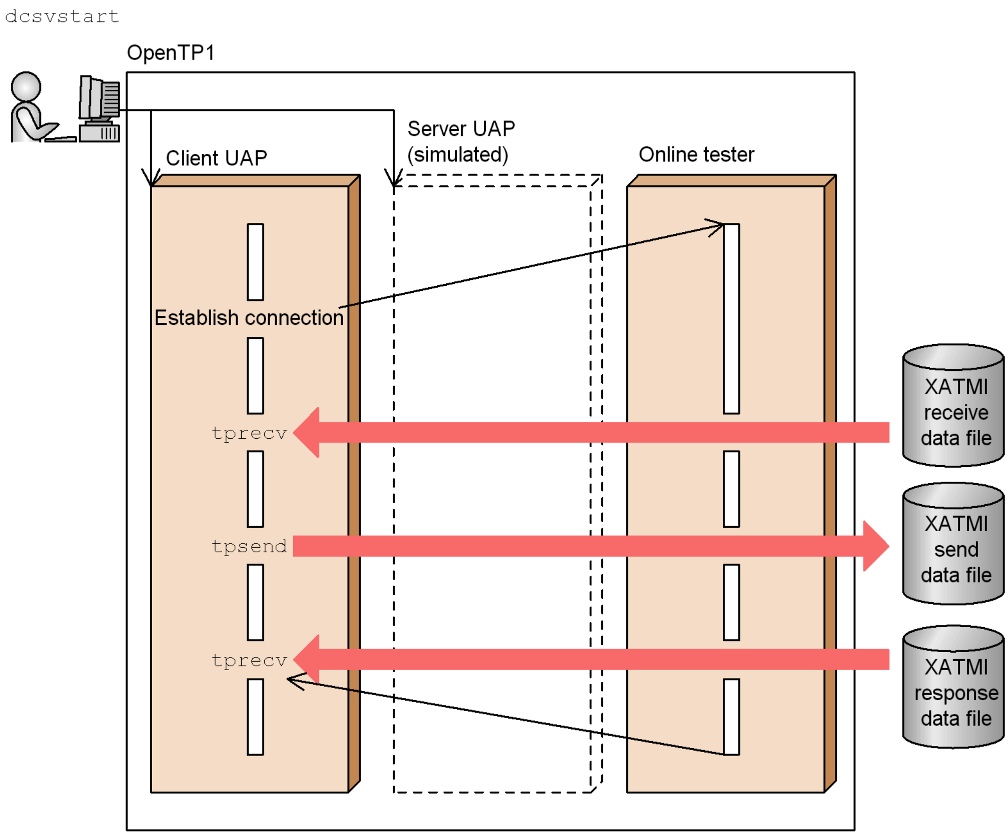
(2) Conversational service paradigm

To simulate a server UAP that accepts the conversational service paradigm, the user must first create an *XATMI receive data file* and XATMI response data file containing the data to be received by the client UAP. When a receive request is sent from the client UAP, the online tester reads an item of receive data from the XATMI receive data file and returns it to the client UAP. If a further receive request is made after all the data in the XATMI receive data file has been returned, the online tester reads response data from the XATMI response data file and returns it to the client UAP.

The data sent by the client UAP is saved to the *XATMI send data file* created by the online tester according to the specification in the user service definition.

Figure 2-6 illustrates the server UAP simulator for the conversational service paradigm.

Figure 2-6: Simulating a server UAP for conversational service paradigm



2.4 Simulating the MCF

The online tester can take the place of the MCF in exchanging messages with an MHP. This allows the user to test the MHP, or the SPP to which the MHP sends service requests, without needing the MCF. This facility is called the *MCF simulator*.

An online tester command is used to start the MHP application. Before executing the command, the user must first create an *MCF receive message file* with the messages to be passed to the MHP. The messages sent from the MHP and SPP are saved to an *MCF send message file* created by the online tester.

Send messages can be edited by online tester command. Also, specific send messages can be recreated in the MCF receive message file and used again.

When an MHP uses the MCF simulator, the online tester manages that MHP. The MHP is not managed by the actual MCF, even if active. Therefore, operating commands provided by the MCF are not available for the MHP.

2.4.1 MCF simulation functions

At execution, the MCF simulator links the MHP to the online tester library rather than to the library provided by the MCF. At linkage to the online tester, the functions used by the MHP are replaced by functions for the online tester. These functions are called *MCF simulation functions*.

To use the MCF simulator, the user must first write a user service definition, defining the MHP for which functions are to be replaced as a *simulate MHP*. A simulate MHP is an MHP that uses MCF simulation functions and runs in test mode (that is, all the facilities of the online tester can be used). A simulate MHP is managed as an SPP by the online tester.

The online tester cannot be used to test a normal MHP (linked to the MCF-supplied library).

2.4.2 Simulating message send/receive

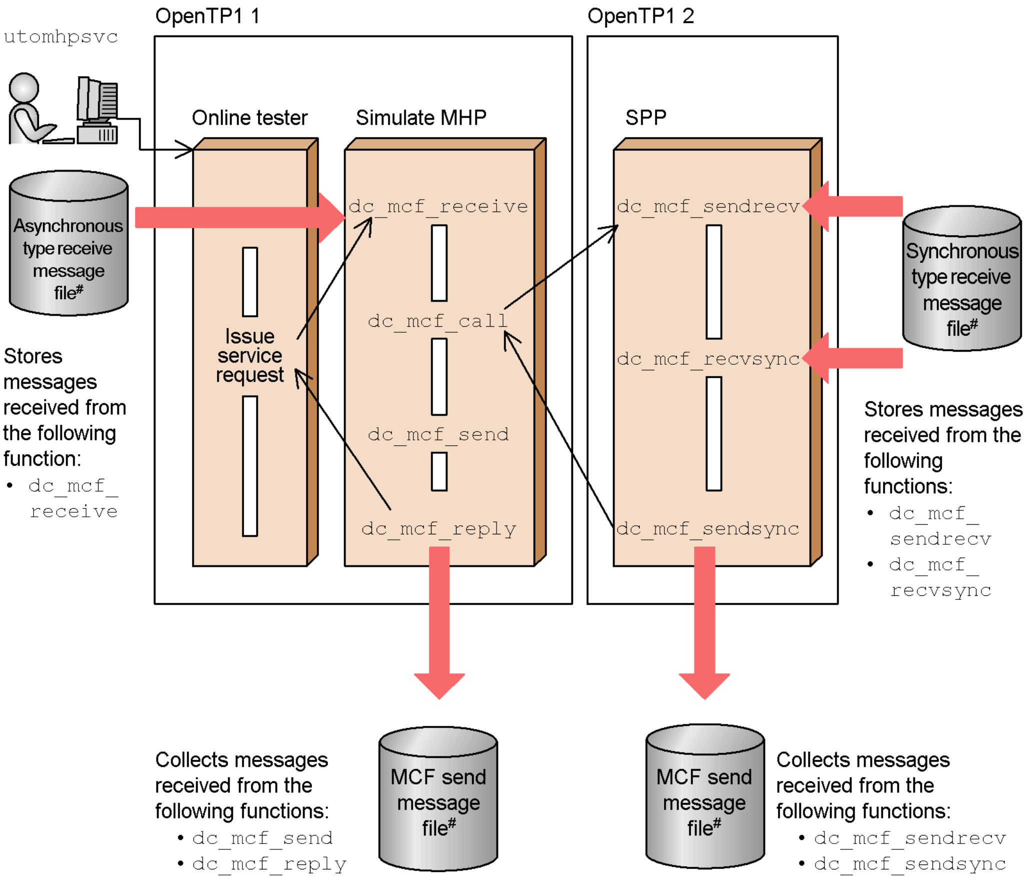
MCF simulation functions simulate message send and receive. Receive messages are created in different MCF receive message files, depending on whether messages are sent and received synchronously or asynchronously.

Asynchronous type receive message files are for simulating asynchronous message send/receive. A single logical message is stored in an asynchronous type receive message file.

Synchronous type receive message files are for simulating synchronous message send/receive. All the logical messages received synchronously during execution of one service are stored in a synchronous type receive message file.

Figure 2-7 outlines simulation of message send/receive.

Figure 2-7: Simulating message send/receive



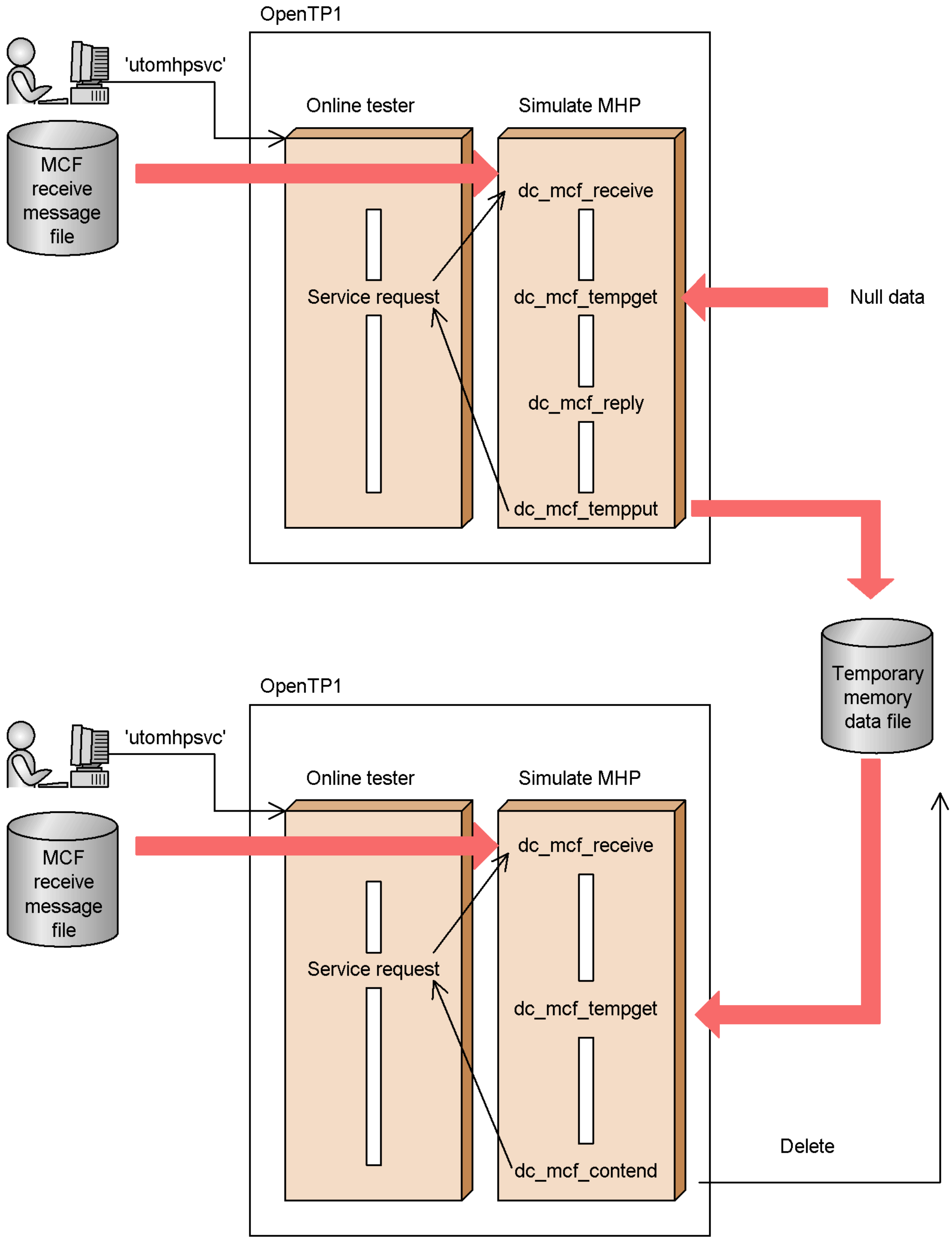
#: MCF receive message files (both asyncronous and synchronous) are created by the user for each test user ID and logical terminal.
MCF send message files are generated by the online tester for each test user ID.

2.4.3 Simulating continuous inquiry responses

Simulation of continuous inquiry responses is executed by online tester command. Temporary memory data is collected in a *temporary memory data file* created by the online tester. This file is automatically deleted when continuous inquiry responses terminate.

Figure 2-8 outlines simulation of continuous inquiry responses.

Figure 2-8: Simulating continuous inquiry responses

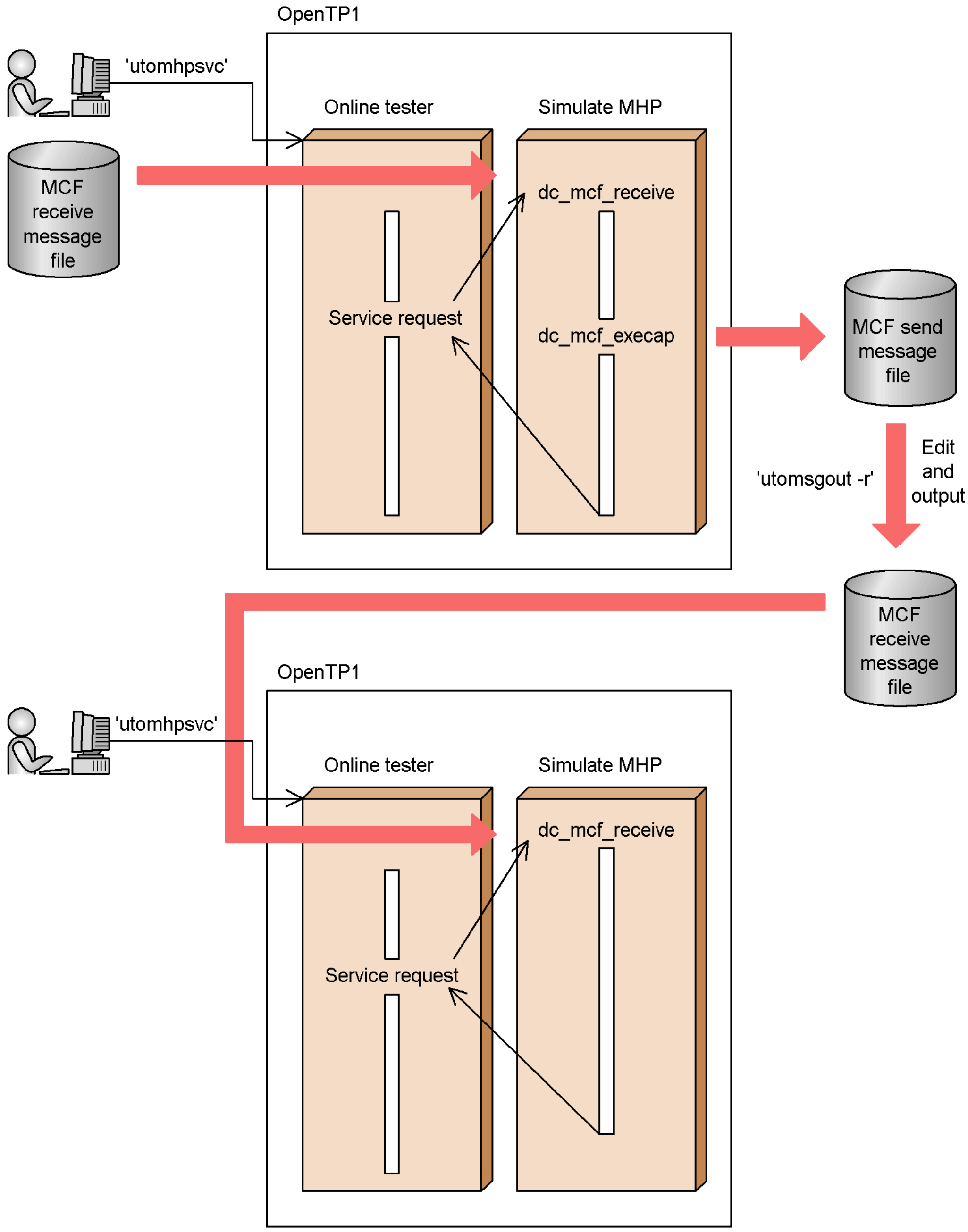


2.4.4 Simulating application program startup requests

Online tester commands can be used to simulate an application program startup request. When a UAP requests startup of an application program, the application program does not actually start, but the data to be passed is saved in an MCF send message file created by the online tester. To send this data and start the application program, the user enters an online tester command to download the data to another file. Then, the user starts the application program by command input, using this file as the MCF receive message file. In this way, a startup request can be simulated for an application program that was not actually started by the UAP.

Figure 2-9 outlines simulation of an application program startup request.

Figure 2-9: Simulating an application program startup request



2.4.5 Simulating synchronous point processing

When a commit request or rollback request is issued by the MHP being tested, the function is actually executed by the online tester. For a commit request, however, the user service definition determines whether a commit or rollback is performed.

Also, even if process termination or re-scheduling occurs during a rollback request, the rollback function is completed and returned.

The online tester cannot handle process termination or re-scheduling. Include such processing within the MHP to be tested.

2.5 Disabling resource updating

The online tester can restore the resources updated during a test. This is called *disabling resource updating*.

Updated resources are restored by rollback at normal termination of the transaction. Whether a commit or rollback is performed at normal termination is determined for the global transaction according to the user service definition for the UAP in which the root transaction branch occurred. When two or more transaction branches occur, the specification for the UAP in which the root transaction branch occurred takes effect, regardless of the specifications for the individual UAPs.

Transaction-dependent journals collected for a transaction being tested can be edited for output in the same way as normal journals, using the `jnledit` OpenTP1 command.

2.6 Simulating operating commands

The online tester can simulate command execution requested by the `dc_adm_call_command` function issued in a UAP. This facility is called the *operating command simulator*.

In the user service definition, the user can specify for each UAP whether to use the operating command simulator. The following two options are available:

(1) Skipping command execution

Operating commands are skipped instead of being executed. The following default information is set as the command execution result (return information of the `dc_adm_call_command` function):

- Shell termination code: 0
- Data output to standard output or standard error output: Null character
- Output data length (standard output or standard error output): 0

(2) Replacing command execution results

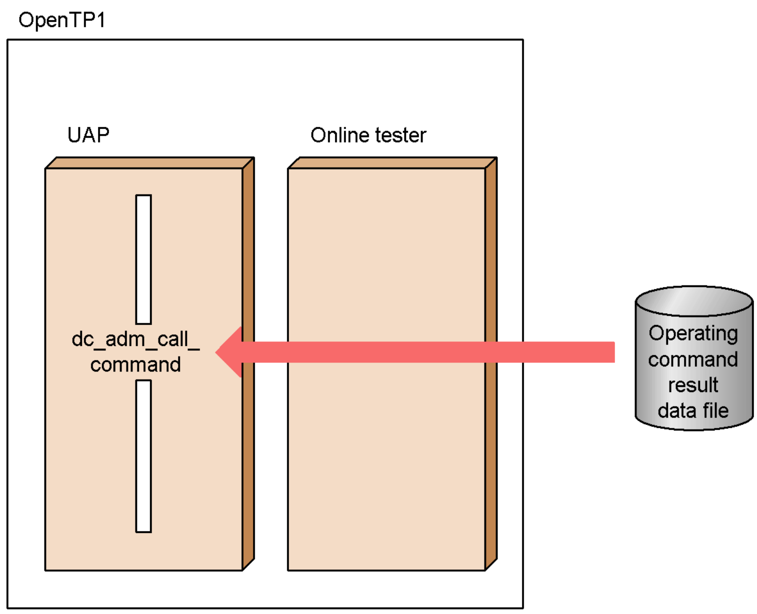
Instead of the operating command being executed, the data in the *operating command result data file* is set as the command execution result. When the UAP issues a `dc_adm_call_command` function, the online tester reads the execution result data from the file and returns the data to the UAP.

An operating command result data file must be created for each service. Set the execution result data in this file before running a test.

If the `dc_adm_call_command` function is issued more than once in a service, the user must create the data to be returned at each function call. This also applies to the main function in the SPP and to functions issued from an SUP.

Figure 2-10 shows how the data in the operating command result data file is used to replace the actual execution result.

Figure 2-10: Replacing command execution results



2.7 Creating and outputting tester files

The online tester uses a number of different simulators, so a dedicated-use data file must be created for each one. These data files are called *tester files*.

This section describes how tester files are created, edited, and output.

2.7.1 Creating tester files

Each tester file is written in a specific data format. However, the user can easily create the tester files by command input, using the online tester. This is called the *tester file creation facility*.

Table 2-1 lists the tester files that can be created with the tester file creation facility.

Table 2-1: Tester files created by tester file creation facility

Tester files		Creator	Simulator using the tester file
Service request data files	RPC request data file	User	Client UAP simulator
	XATMI request data file	User	Client UAP simulator
Service response data files	RPC response data file	User	Server UAP simulator
		Online tester	Client UAP simulator
	XATMI response data file	User	Server UAP simulator
		Online tester	Client UAP simulator
XATMI receive data file		User	Client UAP simulator
MCF receive message files	Asynchronous receive message file	User	MCF simulator
	Synchronous receive message file	User	MCF simulator
Operating command result data file		User	Operating command simulator

The tester file creation facility is used to create tester files with one of following two methods:

- Using the test data definition file

  Creation of a tester file can use data from the test data definition file created by the user. The user can create the test data definition file using a text editor. This file can contain data for multiple tester files.

- Using journal data

  Creation of a tester file can use record data from an unload journal file or trace data from an RPC trace file. To use journal data, extract it using an operating command.

Commands for extracting data and data types depend on kinds of tester files to be created. Using journal data disallows creation of an operating command result data file. Table 2-2 lists the kinds of tester files to be created, corresponding data extraction commands, and available data.

Table 2-2: Kinds of tester files to be created, available data extraction commands, and available data

Tester file name	Data extraction command	Available data
RPC request data file	`rpcdump`	• First effective RPC request send data out of RPC trace data extracted by the `rpcdump` command.
XATMI request data file	`rpcdump`	• First effective `tpcall` or `tpacall` function data out of XATMI request/response request send data in RPC trace data extracted by the `rpcdump` command. • First effective `tpconnect` function data out of XATMI interactive request send data in RPC trace data extracted by the `rpcdump` command.
RPC response data file	`rpcdump`	• First effective RPC request send data out of RPC trace data extracted by the `rpcdump` command.
XATMI response data file	`rpcdump`	• First effective `tpreturn` function data out of XATMI request/response and interactive request send data in RPC trace data extracted by the `rpcdump` command.
XATMI receive data file	`rpcdump`	• All `tprecv` function data out of XATMI interactive receive data in RPC trace data extracted by the `rpcdump` command.
Asynchronous receive message file	`jnlrput`	• ij record data and mj record input message data in unload journal files extracted by the `jnlrput` command. When two or more record data entries are available, the system accepts data entries whose order identifier begins with `s` or `l`.
Synchronous receive message file	`jnlrput`	• Input message data in mj records from unload journal files extracted by the `jnlrput` command.
Operating command result data file	--	--

Legend:

--: No files are created from journal data or trace data.

2.7.2 Editing and outputting tester files

The online tester can edit and output contents of the created tester file. This is called the *tester file edit and output facility*.

To use this facility, execute an online tester command. Entering the command edits data in a specified tester file based on the format of the specified tester file kind and outputs the edited data to the standard output.

Table 2-3 shows tester files available for edit and output with the tester file edit and output facility.

Table 2-3: Tester files available for edit and output with the tester file edit and output facility

<table>
<tr><th colspan="2">Tester files</th><th>Creator</th><th>Simulator using the tester file</th></tr>
<tr><td rowspan="2">Service request data files</td><td>RPC request data file</td><td>User</td><td>Client UAP simulator</td></tr>
<tr><td>XATMI request data file</td><td>User</td><td>Client UAP simulator</td></tr>
<tr><td rowspan="4">Service response data files</td><td rowspan="2">RPC response data file</td><td>User</td><td>Server UAP simulator</td></tr>
<tr><td>Online tester</td><td>Client UAP simulator</td></tr>
<tr><td rowspan="2">XATMI response data file</td><td>User</td><td>Server UAP simulator</td></tr>
<tr><td>Online tester</td><td>Client UAP simulator</td></tr>
<tr><td rowspan="3">XATMI send/ receive data files</td><td rowspan="2">XATMI send data file</td><td rowspan="2">Online tester</td><td>Client UAP simulator</td></tr>
<tr><td>Server UAP simulator</td></tr>
<tr><td>XATMI receive data file</td><td>User</td><td>Client UAP simulator</td></tr>
<tr><td rowspan="2">MCF receive message files</td><td>Asynchronous receive message file</td><td>User</td><td>MCF simulator</td></tr>
<tr><td>Synchronous receive message file</td><td>User</td><td>MCF simulator</td></tr>
<tr><td colspan="2">Operating command result data file</td><td>User</td><td>Operating command simulator</td></tr>
</table>

2.8 Collecting test information

2.8.1 Collecting UAP trace information

The online tester collects UAP trace information for the UAP running in test mode at the entrance and exit of each OpenTP1 function. This is called *collecting UAP trace information*.

The functions provided by OpenTP1 that access a user server, an RPC function, a DAM file, or a TAM file can use an online tester facility that collects a trace of all the I/O data specified in the function. This is called *collecting of the complete I/O data trace*.

Table 2-4 shows the functions that can use an online tester facility that collects the complete I/O data trace.

Table 2-4: Functions that can use the complete I/O data trace collection facility

Facility	**Function name**	
	C format	**COBOL format**
User server	Service function start	Service program start
	Service function end	Service program end
	Service function start (at retry)	Service program start (at retry)
	Service function end (at retry)	Service program end (at retry)
Remote procedure call	`dc_rpc_call`	`CBLDCRPC('CALL    ')`
	`dc_rpc_cltsend`	`CBLDCRPC('CLTSEND ')`
DAM file service	`dc_dam_read`	`CBLDCDAM('DCDAMSVC', 'READ')`
	`dc_dam_rewrite`	`CBLDCDAM('DCDAMSVC', 'REWT')`
	`dc_dam_write`	`CBLDCDAM('DCDAMSVC', 'WRIT')`
TAM file service	`dc_tam_read`	`CBLDCTAM('FxxR')('FxxU')`
	`dc_tam_rewrite`	`CBLDCTAM('MFY ')('MFYS')('STR ')`
	`dc_tam_write`	
IST service	`dc_ist_read`	`CBLDCIST('DCISTSVC', 'READ')`
	`dc_ist_write`	`CBLDCIST('DCISTSVC', 'WRIT')`

Facility	Function name	
	C format	COBOL format
User journal collection	`dc_jnl_ujput`	`CBLDCJNL('UJPUT  ')`

The complete I/O data trace is collected at different times depending on the function, as shown below.

- User server functions

  Collect input data at startup and collect output data at termination.

- Data send/receive functions

  Collect the send data on service requests and the CUP notification data at the start of the function. Also, collect the receive data on service responses at the exit of the function.

- File data read functions

  Collects the trace at the exit of the function.

- File data write functions

  Collects the trace at the start of the function.

UAP trace information is collected in *trace files*. A trace file is created automatically for each OpenTP1 system when the online tester collects the first trace information. When full, the trace file is swapped with another file.

Trace information for a number of OpenTP1 functions is collected in one file at completion of a service function, for example. Also, if the UAP terminates abnormally, trace information is extracted from the core file and saved in the trace file. For this reason, trace information may not be collected if the online tester is immediately terminated during UAP execution or if no core file is collected at abnormal UAP termination.

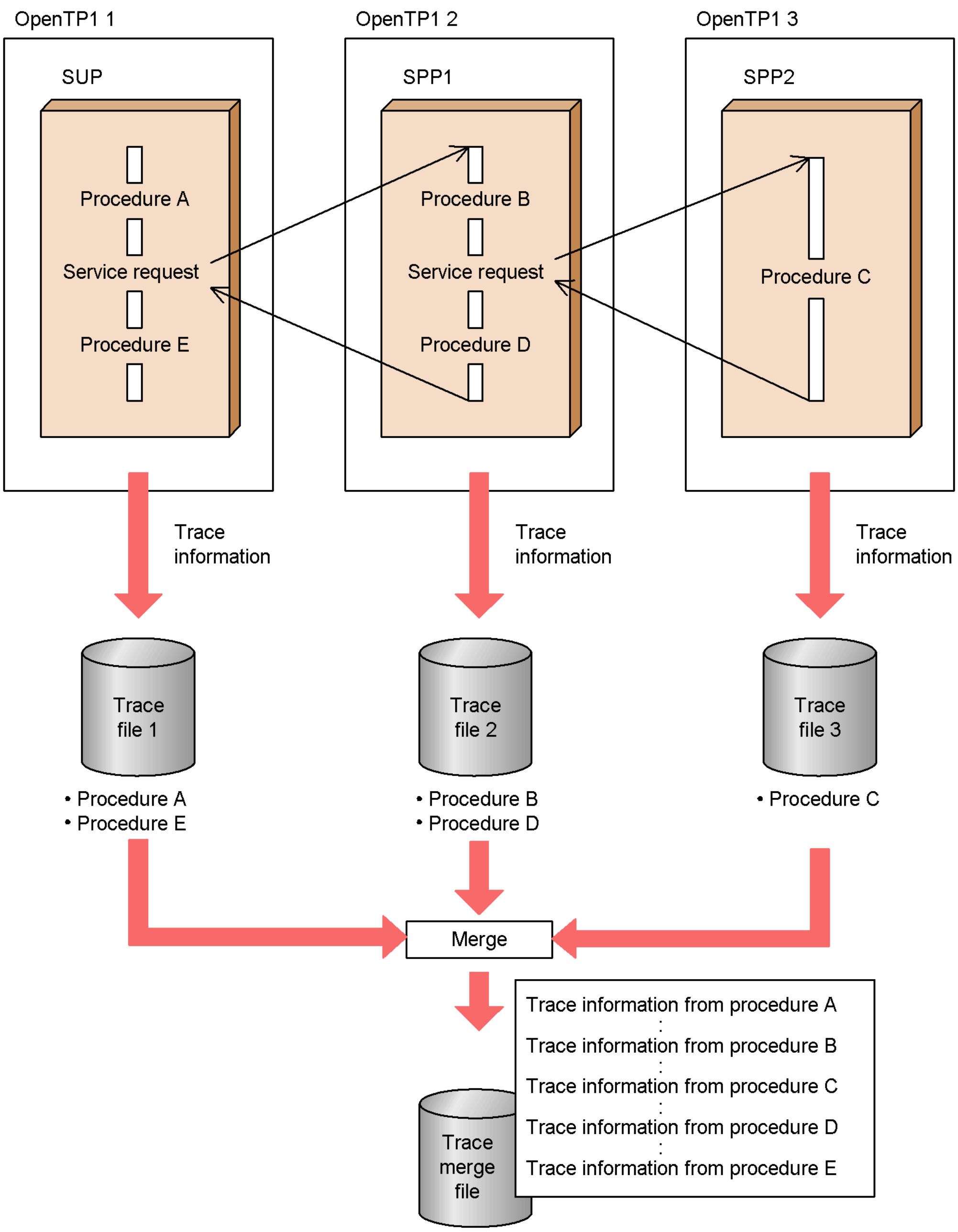
2.8.2 Merging, editing, and outputting UAP trace information

The online tester can merge UAP trace information from a number of trace files into a single file and edit the file contents for output. This is called *merging and editing UAP trace information*.

Trace information is merged by entering an online tester command. The user specifies two or more trace files and the trace information is merged in a single file, following the service sequence. This facility can be used for saving the trace information from a number of OpenTP1 systems in collection sequence for each global transaction. The facility can also be used for merging the contents of a trace file and swap file.

Figure 2-11 shows the result of merging UAP trace information.

Figure 2-11: Result of merging UAP trace information



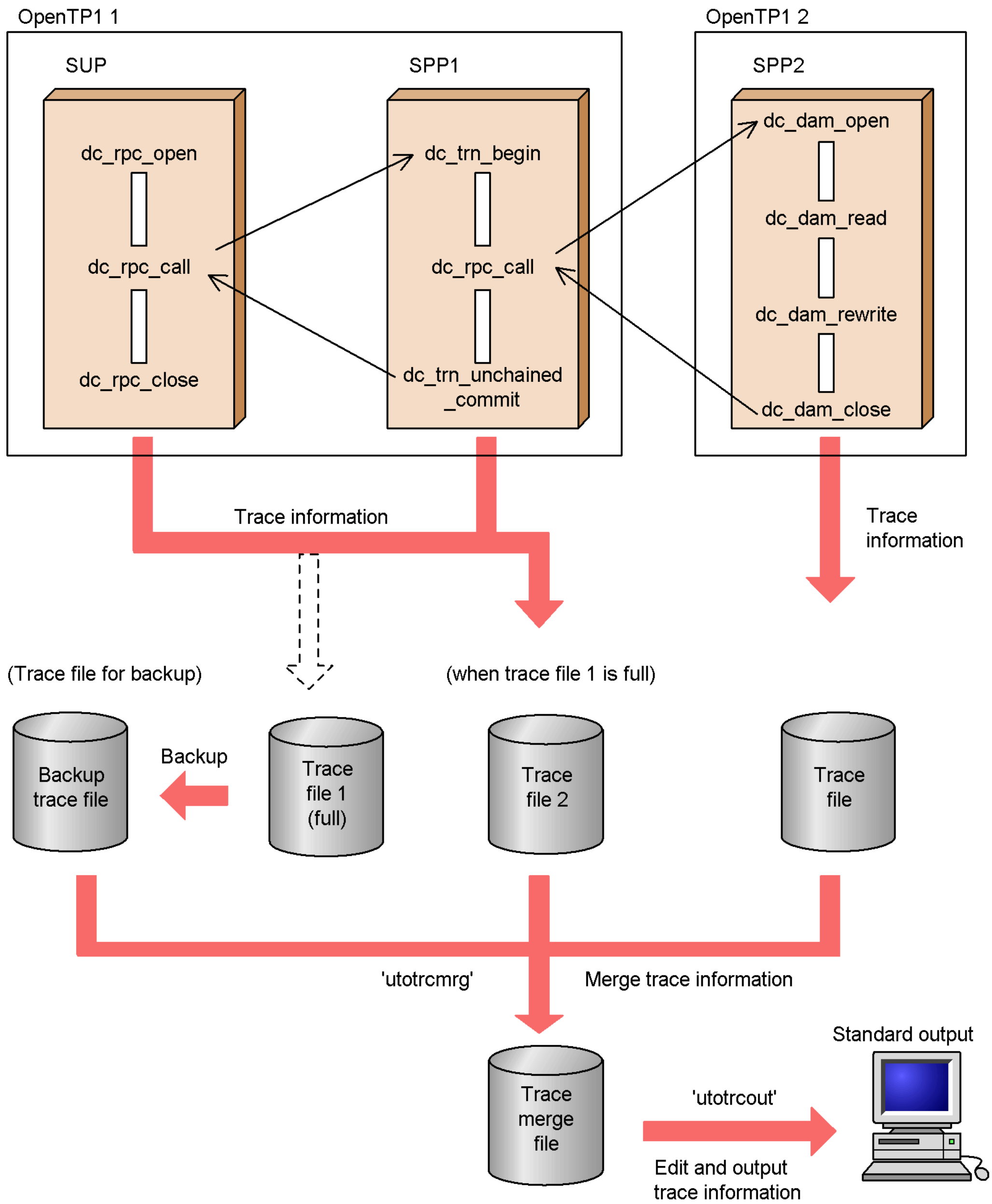
Trace information can be edited for output by online tester command. The user can specify the log date and time to set the output range.

Two output formats are available:

- All the trace information in a trace file
- Part of the trace information (function names, for example) in a trace file

Figure 2-12 gives an overview of collecting, merging, editing, and outputting UAP trace information.

Figure 2-12: Collecting, merging, editing, and outputting UAP trace information



2.8.3 Editing send messages

The send messages collected in the MCF send message file when using the MCF simulator can be edited for output. This is called *editing send messages*.

Send messages are edited by entering an online tester command. The data in the MCF send message file is edited and output to the file specified in the command or to standard output.

2.9 Interlocking the debugger

The online tester can online test test-only UAPs such as SUP, SPP, and MHP by interlocking with the debugger. This facility is called *debugger interlocking*.

Specify debugger interlocking for each UAP in the user service definition and execute the online tester command. The main function in the UAP activates the debugger. Interlocking the debugger easily provides step-by-step debugging or batch debugging.

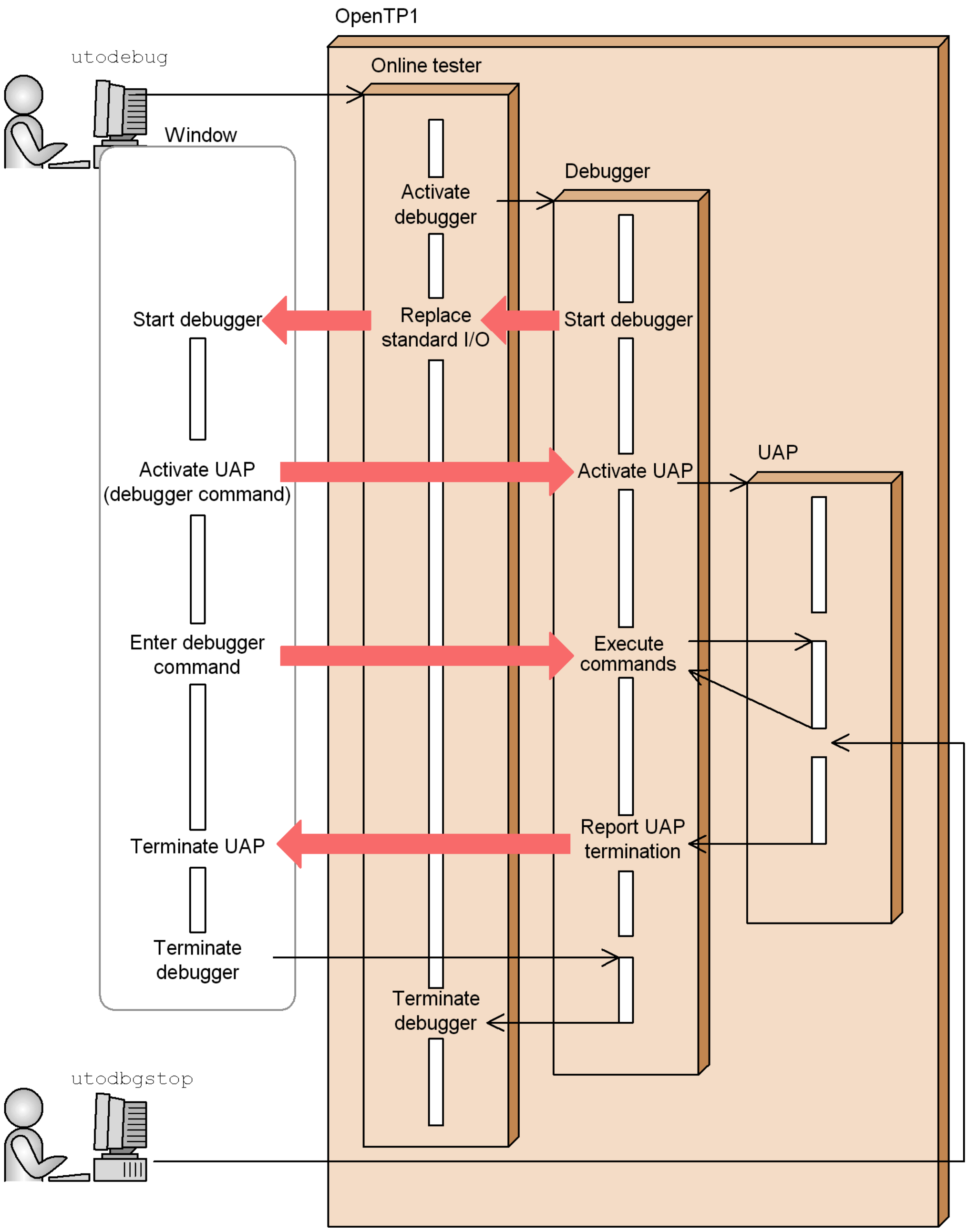
The available debugger is:

- `dbx`
- `cbltd (COBOL85/TD)`
- `cblcv (COBOL85/TD)`

Before testing a UAP online by interlocking the debugger, test that UAP is offline. Do not let more than one user interlock the debugger on the same node to perform a test. This is a guard against an effect on the OpenTP1 system when the UAP interlocked to the debugger terminates abnormally.

The user can collect trace information about a UAP interacting with the debugger in the same manner as a UAP that operates independently of the debugger. However, part or all of the trace information may be unavailable depending on the timing when the online tester writes trace information. Figure 2-13 outlines how debugger interlocking works.

Figure 2-13: Interlocking the debugger



Chapter

3. Setting the Test Environment

This chapter explains how to set the environment for running tests with the online tester.

This chapter contains the following sections:

3.1 System definitions for the online tester
3.2 Setting environment variables
3.3 User-created files
3.4 Creating files

3.1 System definitions for the online tester

The system definitions for running the online tester are described below. See the manual *OpenTP1 System Definition* for information on definition structure and rules.

3.1.1 System service configuration definition

Add the following definition to the OpenTP1 system service configuration definition (definition file name: `$DCCONFPATH/sysconf`).

(1) Syntax

(a) set format

```
[set uto_conf=Y|N]
```

(b) Command format

None.

(2) Function

Defines whether to start the online tester at system startup.

(3) Explanation

(a) set format

Operands

- `uto_conf=Y|N` ~<<N>>

  Specify whether to use the online tester at this node.

  `Y`

    Use the online tester.

  `N`

    Do not use the online tester.

(b) Command format

None.

3.1.2 Tester service definition

Create the definition file `$DCCONFPATH/uto`, then define in this file the tester service definition.

(1) Syntax

(a) set format

```
[set uto_server_count=maximum no. of test user servers]
[set max_trace_file_size=maximum size of trace files]
[set max_message_file_size=maximum size of MCF send
                           message file]
[set watch_time=max reply wait time]
[set rpc_trace=Y|N]
[set rpc_trace_name="name of the file for collecting RPC
                     trace information"]
[set rpc_trace_size=size of the file for collecting RPC
                    trace information]
```

(b) Command format

```
{{[<INDEXWORD PRONOUNCE="utoterm" INDEXITEM="utoterm" PARENTPRONOUNCE="Tester Service
Definition" PARENTITEM="Tester Service Definition">utoterm</INDEXWORD> [-p
OSITP|other] logical-terminal-name]}}
```

(2) Function

Defines the environment for executing online tester services.

(3) Explanation

(a) set format

Operands

- `uto_server_count` ~<unsigned integer> ((0-240)) <<64>>

  Specify the maximum number of user servers that can be activated for testing by the online tester.

- `max_trace_file_size` ~<unsigned integer> ((0-2000000)) <<64>> (unit: Kbytes)

  Specify the maximum size of each trace file for storing UAP trace information. As the header (management information) in a trace file is 128 bytes, add 128 bytes to the value you wish to specify.

  When zero is specified, the online tester does not collect UAP trace information.

  Up to two trace files are created for each tester user ID. This prevents erasure of the trace information when a trace file becomes full.

  Whenever a trace file becomes full, the trace information must be backed up, and then the full trace file is deleted. Prevent the trace file from becoming full by specifying sufficient size.

  The maximum trace file size that can be specified for an OpenTP1system is given by the following equation:

Maximum size of trace file =

(value specified in max_trace_file_size operand) x 2 x (number of users) x 1,024 bytes

- max_message_file_size ~<unsigned integer> ((0-2000000)) <<64>> (unit: Kbytes)

  Specify the maximum size of the MCF send message file for storing messages sent by the following functions when the online tester's MCF simulator is used:

  - dc_mcf_send function
  - dc_mcf_reply function
  - dc_mcf_sendsync function
  - dc_mcf_sendrecv function
  - dc_mcf_execap function

  As the management information data in each send message is 128 bytes, and the header (management information) in the MCF send message file is 128 bytes, add 128 bytes to the value you wish to specify.

  When zero is specified, the online tester does not collect send messages.

  An MCF send message file is created for each test user ID. The maximum size that can be specified for a MCF send message file in an OpenTP1 system is given by the following equation:

  Maximum size of MCF send message file =

  (value specified in max_message_file_size operand) x (number of users) x 1,024 bytes

- watch_time ~<unsigned integer> ((0-62535)) (unit: seconds)

  When using remote procedure calls (RPCs) for inter-process communication, specify the maximum wait time for return of a service reply after a service request is sent.

  OpenTP1 may suspend termination processing for the length of time specified in this operand. Therefore, if you specify a large value, the termination processing of OpenTP1 may take some time.

  If no reply has been received when the specified time elapses, the RPC returns a send/receive timeout error.

  When zero is specified, the system remains in wait state indefinitely.

  When you specify zero, OpenTP1 may not terminate.

When specification is omitted, the value specified in the `watch_time` operand of the system common definition is assumed.

- `rpc_trace=Y|N` ~<<N>>

  Specify whether to collect RPC traces.

  `Y`

  Collect RPC traces.

  `N`

  Do not collect RPC traces.

  When specification is omitted, the value specified in the `rpc_trace` operand in the system common definition is assumed.

- `rpc_trace_name` ~<pathname> `<<$DCDIR/spool/rpctr>>`

  Specify the pathname of the file for collecting RPC traces.

  In the pathname, the maximum length of the name of the file for acquiring the RPC trace is 13 characters. The default file name is `rpctr`.

  To specify an environment variable in a pathname, make sure that the pathname begins with the environment variable (example: `$DCDIR/tmp/`*file-name*).

  When specification is omitted, the value specified in the `rpc_trace_name` operand in the system common definition is assumed.

- `rpc_trace_size` ~<unsigned integer> ((1024-2147483648)) <<4096>> (Unit: bytes)

  Specify the size of the file for collecting RPC traces.

  When specification is omitted, the value specified in the `rpc_trace_size` operand in the system common definition is assumed.

(b) Command format

See below.

3.1.3 Tester service definition (command format)

(1) utoterm (specification of logical terminal information)

Syntax

```
{{[utoterm [-p OSITP|other] logical-terminal-name]}}
```

Function

Defines information for each logical terminal when using the MCF simulator for testing an MHP created in the data manipulation language (DML).

When a name already specified is respecified as the logical terminal name, a warning message is displayed and the repeat specification is ignored.

Options

- `-p OSITP|other` ~<<other>>

  Specify the protocol type. Specify this option when testing an MHP created in the DML.

  `OSITP`

  OSI TP protocol

  `other`

  Protocol other than OSI TP

- *logical-terminal-name* ~<identifier of 1-8 characters>

  Specify the logical terminal name.

3.1.4 User service definition

Add the following definitions to the OpenTP1 user service definition (definition file name: `$DCCONFPATH/`*user-server-name*).

(1) Syntax

(a) set format

```
[set test_mode=target|usable|dmyspp|simmhp|no]
[set test_transaction_commit=Y|N]
[set test_adm_call_command=do|skip|file]
[set test_xatmi_send_file=Y|N]
[set test_debugger="{dbx|cbltd|cblcv}[command-argument]"]
[set test_data_trace=Y|N]
```

(b) Command format

None.

(2) Function

Enables execution of the online tester at the user server. Add the definitions to each service group in the OpenTP1 user service definition.

(3) Explanation

(a) set format

Operands

- `test_mode=target|usable|dmyspp|simmhp|no` ~<<no>>

  Specify whether the UAP is to be tested when the online tester is activated.

`target`

Test-only UAP

Specify this option to set the UAP as a test-only UAP. All the facilities of the online tester (disabling resources updating, collecting UAP trace information, and so on) are used in testing the UAP.

Service requests cannot be made from a test-only UAP to a non-test UAP, or from a non-test UAP to a test-only UAP.

`usable`

Usable UAP

Specify this option to set the UAP as an SPP to which service requests are sent from the UAP being tested.

A usable UAP runs in test mode when the UAP being tested makes a service request. The facilities of the online tester, such as disabling resources updating, can be used.

When a service request is made from a non-test UAP, the usable UAP runs in non-test mode and the online tester facilities are not available.

`dmyspp`

Dummy SPP

Specify this option to use the online tester's server UAP simulator to simulate the SPP without actually activating it.

`simmhp`

Simulate MHP

Specify this option to use the online tester's MCF simulator and link simulation functions to the MHP.

`no`

Non-test UAP

Specify this option to exclude the UAP from testing. Service requests cannot be made from a test UAP to a UAP with the `no` specification.

The following tables show the relationships between the `test_mode` operands and the online tester facilities that can be used for UAPs, as well as the relationships between a calling UAP and a called UAP when a service is requested.

Table 3-1: test_mode specifications and available test facilities

Available test facility	target	usable	dmyspp	simmhp	no
Client UAP simulator	Y	Y	--	--	N
Server UAP simulator	Y	Y/N	--	Y	N
MCF simulator	Y#	Y/N	--	Y#	N
Disabling the resources update process	Y	Y/N	--	Y	N
Operating command simulator	Y	Y/N	--	Y	N
Collecting UAP trace information	Y	Y/N	--	Y	N
Debugger interlocking	Y	N	--	Y	N

Legend:

Y: Available.

Y/N: May be available, depending on the type of function.

Main function

Not available.

Service function

Available when using the client UAP simulator for service requests. In other cases, the test facility is available if it can be used with the calling UAP (or with the UAP that makes the first request when a service extends over multiple UAPs).

N: Not available.

--: Not applicable.

#: The UAP must be linked to the MCF simulation functions provided by the online tester.

Table 3-2: Relationships between calling UAP and called UAP when requesting services

Calling UAP		Called UAP				
		target	usable	dmyspp	simmhp	no
`target`		Y	Y	Y	--	N
`usable`	Test mode	Y	Y	Y	--	N
	Non-test mode	N	Y	N	--	Y

Calling UAP	Called UAP				
	target	**usable**	**dmyspp**	**simmhp**	**no**
`dmyspp`	--	--	--	--	--
`simmhp`	Y	Y	Y	--	N
`no`	N	Y	N	--	Y

Legend:

Y: Service requests can be made.

N: Service requests cannot be made.

--: Not applicable.

- `test_transaction_commit=Y|N` ~<<N>>

  Specify whether a commit or rollback is performed at a synchronous point when a transaction running in test mode occurs in this UAP.

  `Y`

  Commit

  `N`

  Rollback

- `test_adm_call_command=do|skip|file` ~<<do>>

  Specify whether to simulate operating command execution when a `dc_adm_call_command` function is issued in this UAP.

  `do`

  Execute operating commands.

  `skip`

  Instead of executing the command, use the default result.

  This option is valid only when `target` or `simmhp` is set in the `test_mode` operand, or when `usable` is specified and the UAP is running in test mode.

  `file`

  Instead of executing the command, use the data in the operating command result data file as the execution result.

  This option is valid only when `target` or `simmhp` is set in the `test_mode` operand, or when `usable` is specified and the UAP is running in test mode.

- `test_xatmi_send_file=Y|N` ~<<N>>

  Specify whether the data sent to the simulated UAP by the server UAP simulator is to be output to the XATMI send data file when a conversational service is requested in an XATMI interface.

  `Y`

  Outputs the send data to the file.

  `N`

  Does not output the send data to the file.

  This option is ignored if specified for a UAP other than a simulated UAP (`dmyspp` specified in the `test_mode` operand).

- `test_debugger="{dbx|cbltd|cblcv}[`*command-argument*`]"`

  When activating the UAP by interlocking the debugger, specify the necessary debugger command name and a command argument for that debugger command.

  When a UAP is given this definition, executing the `utodebug` command activates this UAP together with the specified debugger.

  Inadvertently executing the `dcsvstart` or `dcstart` command for a UAP with this definition causes the command to fail, outputting an error message.

  To terminate the UAP that was activated with the `utodebug` command, execute the `utodbgstop` command from a window other than that was used to execute the `utodebug` command.

  If the UAP is terminated with a command other than the `utodbgstop` command, the OpenTP1 system and the online tester may provide different UAP states. The executed command must wait until the debugger terminates.

  It is impossible to re-activate a UAP process that was activated with the debugger. After termination of the UAP process that is interlocked to the debugger, reexecuting this UAP needs to stop the debugger, then reexecute the `utodebug` command.

  The UAP with this definition can be active in a single process regardless of the `parallel_count` operand value specified in the user service definition of the corresponding UAP.

  It is impossible to enable or disable shutdown for the UAP that was activated with the debugger.

- `test_data_trace=Y|N` ~<<N>>

  Specify whether the complete I/O data issued in this UAP for the function is to be collected as trace information. For the function that can use a facility that collects the complete I/O data trace information, see Subsection *2.8.1 Collecting UAP* *trace information*.

`Y`

Collects the complete I/O data as UAP trace information.

This option is valid only when a value of 1 or greater is specified in the `max_trace_file_size` operand of the tester service definition to use the UAP as the test target.

`N`

Collects a part of the I/O data as UAP trace information.

This option is valid only when a value of 1 or greater is specified in the `max_trace_file_size` operand of the tester service definition to use a UAP as the test target.

(b) Command format

None.

(4) Notes

- When specifying `simmhp` in the `test_mode` operand, match all the other specifications in the user service definition with the SPP specifications.

  Example:

```
type=other
```

  Also, specify `queue` in the `receive_from` operand.

- User service default definitions cannot be specified as online tester definitions in the user service definition. This prevents a real job UAP from being run in error in a test environment.
- When using the MCF simulator, specify `Y` in the `atomic_update` operand of the user service definition if a transaction MHP is to be executed.
- The schedule priority of a test UAP depends on value specified in the `schedule_priority` operand of the user service definition. When executing a test UAP concurrently with a real job UAP, consider the effect on the performance of the job UAP when specifying the priority of the test UAP.
- If zero is specified in the `uap_trace_max` operand of the user service definition (even if a value of 1 or higher is specified in the `max_trace_file_size` operand of the tester service definition), a warning message output, indicating that UAP trace information cannot be collected.
- When the online tester is not being used, the `dc_rpc_open` function returns an error code at activation of a UAP that has a value other than `no` specified in the

`test_mode` operand. This prevents a test UAP from being run in error as a real job UAP.

- To interlock a UAP to the debugger, specify `file` or `skip` for the `test_adm_call_command` operand. When the UAP is interlocked to the debugger by specifying `do` for the `test_adm_call_command` operand, issuing the `dc_adm_call_command` function lets the UAP wait for a response, disabling debugger control. To solve this, issue the `utodbgstop` command to terminate the UAP, then terminate the debugger.
- Do not issue a `fork` system call or `system(3C)` function to a UAP interlocked with the debugger. Issuing these functions lets the UAP wait for a response, disabling debugger control. To solve this, issue the `utodbgstop` command to terminate the UAP, then terminate the debugger.
- Debugger interlocking is unavailable when running under the multi-node environment.
- If possible, avoid testing a UAP interlocked with the debugger under the OpenTP1 system where a real job UAP is active. This is to prevent a system failure caused by normal or abnormal termination of the UAP interlocked with the debugger under the OpenTP1 system where the UAP is operating.
- Consider debugger interlocking operations and operation times when specifying monitoring time values in the user service definition for the UAP interlocked with the debugger. A thoughtlessly specified value may frequently cause a time-out error.
- When `do` is specified for the `test_adm_call_command` operand for a UAP to start another UAP in the test mode using the `dcsvstart` command set in the argument of the `dc_adm_call_command` function, specify the environment variable `DCUTOKEY` in the user service definition of the UAP that issues the function.

3.1.5 Setting the typed buffer

Typed buffer information must be set to simulate a UAP that uses the XATMI interface. Typed buffer information is stored in a *typed buffer definition file* (any file name).

(1) Syntax

```
zueng020.tif0type zueng020.tif1subtype zueng020.tif1buffer-length
```

Legend:

Δ_0

One or more spaces or tab codes (or none)

Δ_1

One or more spaces or tab codes

(2) Operands

- *type* ~<8 upper-case alphabetics>

  Specify either of the following buffer types:

  - `X_COMMON`
  - `X_C_TYPE`

- *subtype* ~<1-16 alphanumerics>

  Specify the buffer subtype. When the specification exceeds 16 characters, only the first 16 are valid.

  Up to 512 subtypes can be defined for `X_COMMON` or `X_C_TYPE`. When more than 512 subtypes are defined, an error occurs and the `utoxsppsvc` command is terminated.

  When a subtype is duplicated, the first definition is valid. The second and subsequent definitions result in an error and an error message is output. No error message is output, however, when identical contents are defined for the duplicated subtypes.

- *buffer-length* ~<decimal digit>

  Specify the buffer length. Check the buffer length by referring to the stub source created by the TP1/Server Base `stbmake` command and an output result created by the `stbmake` command with the `-p` option specified.

(3) Definition example

```
# typed-buffer-definition
X_COMMON subtype1 256
X_COMMON subtype2 128
X_C_TYPE subtype3 128
#
```

(4) Notes

- Specify one subtype name per line.
- A line can be up to 512 bytes in length, including the line feed code.
- Write # at the start of a comment. Only a space or tab code may be written before #.

  Do not write a comment at the end of the typed buffer definition.

- No error occurs for the file when no valid definitions exist. However, an error occurs when the typed buffer is allocated when, for example, a service is requested.

3.1.6 Setting send/receive procedures

Send/receive procedures must be set when using the conversational service paradigm with a simulated UAP that uses the XATMI interface. Send/receive procedures are stored in a *send/receive control file* (any file name).

In the send/receive control file, define a `send` statement for sending data to the test SPP and a `recv` statement for receiving data from the test SPP.

Always create a send/receive control file when using the conversational service paradigm, even if no data is actually sent or received.

(1) Syntax

(a) send statement

```
zueng020.tif0send [zueng020.tif1XATMI-receive-data-file-name]
```

Legend:

Δ_0

One or more spaces or tab codes (or none)

Δ_1

One or more spaces or tab codes

(b) recv statement

```
Δ0recv Δ1type Δ1subtype or buffer-length [Δ1flag [,flag...]]
```

Legend:

Δ_0

One or more spaces or tab codes (or none)

Δ_1

One or more spaces or tab codes

(2) Operands

(a) send statement

- `send`

Specify the `send` keyword as the definition name.

- *XATMI-receive-data-file-name* ~<pathname>

  Specify the name of the XATMI receive data file containing the data received by the test SPP.

  When specification is omitted, the data in the XATMI receive data file specified in the preceding `send` statement is used. An error occurs if specification is omitted for the first `send` statement.

(b) recv statement

- `recv`

  Specify the `recv` keyword as the definition name.

- *type* ~<8 upper-case alphabetics>

  Specify one of the following receive buffer types:

  - `X_OCTET`
  - `X_COMMON`
  - `X_C_TYPE`

- *subtype* ~<1-31 alphanumerics>

  Specify the receive buffer subtype when specifying `X_COMMON` or `X_C_TYPE` in *type*.

- *buffer-length* ~<decimal digit>

  Specify the receive buffer length when specifying `X_OCTET` in *type*.

- *flag*

  Specify one or more of the following flags set for a receive request (`tprecv` function):

  - `TPNOCHANGE`
  - `TPNOBLOCK`
  - `TPNOTIME`
  - `TPSIGRSTRT`

  Do not specify a flag unless required.

  When setting multiple flags, delimit each flag with a comma (,). Do not insert a space or tab code before or after the comma.

(3) Definition example

```
# send/receive procedure definition
# interactive service name: service01
send sendfile1
recv X_OCTET 128 TPNOCHANGE
send sendfile2
recv X_COMMON subtype1 TPNOTIME,TPSIGRSTRT
#
```

(4) Notes

- A line can be up to 512 bytes in length, including the line feed code.
- Write # at the start of a comment. Only a space or tab code may be written before #.

  Do not write a comment at the end of the send/receive procedure definition.
- No error occurs if no `send` statement or `receive` statement is defined. However, processing is terminated if a connection is established during execution of the `utoxsppsvc` command.
- During execution of the `utoxsppsvc` command, the `tpsend` and `tprecv` functions are issued for the conversational service paradigm according to the specifications in the send/receive control file. If a `TPEV_SVCSUCC` or `TPEV_SVCFAIL` event occurs, subsequent `send` and `recv` statements are ignored and the command terminates normally.
- The definition in the `recv` statement is related to the XATMI functions issued by the `utoxsppsvc` command as follows.

  Example:

  If the `recv` statement is defined as: `recv X_COMMON subtype1 TPNOCHANGE`

  Then:

```
ptr=tpalloc(X_COMMON, subtype1,0)
              1.          2.
tprecv(cd,&ptr,&len,TPNOCHANGE,&revent);
                        3.
```

1. type
2. subtype
3. flag

3.2 Setting environment variables

If two or more users run tests on the same OpenTP1 system, the trace information may be mixed and the test results may be difficult to verify. To prevent this risk, a *test user ID* is set for each user of the online tester. The online tester assigns output files for trace information and MCF send messages, using the test user IDs.

Set a unique test user ID for each user, subject to the following conditions:

Environment variable	Value attribute	Number of characters
DCUTOKEY	1-byte alphanumerics (a-z, A-Z, and 0-9)	Up to 4

Setting test user IDs means that trace files and MCF send message files can be created and used by each test user ID.

Test user IDs are obtained at the following times:

- At UAP startup by the OpenTP1 `dcsvstart` command
- At specification of the `dcsvstart` command in the user service configuration definition
- At a service request to an SPP by the online tester's `utosppsvc` or `utoxsppsvc` command
- At a service request to an MHP by the online tester's `utomhpsvc` command

The test user ID may be assumed as `_uto` when the OpenTP1 system is restarted after forced termination (`-f` option in the `dcsvstop` command) or after a system shutdown during normal termination processing of a test UAP. A message is output, reporting that the system was restarted with the assumed test user ID. If necessary, re-enter the `dcsvstart` command to restart the OpenTP1 system after UAP termination.

3.3 User-created files

The following tables list the types and names of the tester files that the user must create in order to use the online tester.

For creating a test directory, see Subsection *3.4.1 Test directory*.

Table 3-3: List of tester files to be created by the user

Tester file type		Use and contents	Time of creation	Deleted by	Time of deletion
Service request data files	RPC request data file	Stores request data passed to the server UAP when using the client UAP simulator with an RPC interface.	Before service request	User	Any
	XATMI request data file	Stores request data passed to the server UAP when using the client UAP simulator with an XATMI interface.	Before service request	User	Any
Service response data files	RPC response data file	Stores data returned as the service result when using the server UAP simulator with an RPC interface.	At activation of the simulate SPP	User	Any
	XATMI response data file	Stores data returned as the service result when using the server UAP simulator with an XATMI interface.	At activation of the simulate SPP	User	Any
XATMI receive data file		Stores data received by the `tprecv` function in the UAP when the conversational service paradigm is made via an XATMI interface.	Before service request or at activation of the simulate SPP#	User	Any
MCF receive message files	Asynchronous receive message file	Stores messages passed to the MHP by the `dc_mcf_receive` function when using the MCF simulator.	Before service request	User	Any
	Synchronous receive message file	Stores messages passed to the UAP by the `dc_mcf_recvsync` and `dc_mcf_sendrecv` functions when using the MCF simulator.	Before service request	User	Any
Operating command result data file		Stores data returned to the UAP as the execution result when using the operating command simulator.	Before service request	User	Any

Note

All user-created files for the offline tester can be used without modification, except the following:

XATMI receive data file

Synchronous receive message file

Operating command result data file

However, these three files can be used by the offline tester if you use the `cat` command to consolidate several offline tester data files into a single file.

#: The user creates an XATMI receive data file before a service request is made when using the client UAP simulator, or at activation of the simulate SPP when using the server UAP simulator.

Table 3-4: Names for user-created tester files

Tester file type		File name
Service request data files	RPC request data file	Any
	XATMI request data file	
Service response data files	RPC response data file	`$DCDIR/spool/uto/`*test-user-ID*`/` *user-server-name*`/`*svc-service-name*[#1]
	XATMI response data file	`$DCDIR/spool/uto/`*test-user-ID*`/` *user-server-name*`/`*xsv-service-name*[#1]
XATMI receive data file		`$DCDIR/spool/uto/`*test-user-ID*`/` *user-server-name*`/`*xrv-service-name*[#1, #2, #3]
MCF receive message files	Asynchronous receive message file	`$DCDIR/spool/uto/`*test-user-ID*`/` *xx....xx*(*xx....xx* can be any name)
	Synchronous receive message file	`$DCDIR/spool/uto/`*test-user-ID*`/` *recv-logical-terminal-name*[#4] (Header segment file: `$DCDIR/spool/uto` `/`*test-user-ID*`/`*recvh-logical-terminal-name*)
Operating command result data files	For SPP service functions	`$DCDIR/spool/uto/`*test-user-ID*`/` *user-server-name*`/`*cmd-service-name*[#1]
	For SUP and SPP main functions	`$DCDIR/spool/uto/`*test-user-ID*`/` *user-server-name*`/`*cmd*

#1: When the service name exceeds 11 characters, the first five and last six characters are combined as the service name.

Example:*Service name* `uapservice0001` ➔ `uapsece0001`

#2: When the service name exceeds 15 characters, the first five and the 10th to 15th characters are combined as the service name.

Example:*Service name* `uapxatmiservice0001` ➔ `uapxaervice`

#3: Any name when using the client UAP simulator.

#4: Logical terminal name set as the argument of the `dc_mcf_recvsync` or `dc_mcf_sendrecv` function.

3.3.1 Service request data files

(1) RPC request data file

An RPC request data file stores the data passed to the service function for the service specified by the `utosppsvc` command when using the client UAP simulator with an RPC interface. A single file contains one set of data.

(a) File structure

Data length	Response area length	Data

(b) File contents

Item	Position	Length (bytes)	Contents
Data length	0	4	Length of the data to be passed to the service function. (1 to specified value of `DCRPC_MAX_MESSAGE_SIZE`)
Response area length	4	4	Length of the response area to be passed to the service function. (1 to specified value of `DCRPC_MAX_MESSAGE_SIZE`)
Data	8	*n*	Data to be passed to the service function.

(c) Notes

- The items in the RPC request data file are related to the service function arguments as follows:

$$\text{Service function } (\underbrace{\texttt{in}}_{1.},\underbrace{\texttt{in\_len}}_{2.},\texttt{out},\underbrace{\texttt{out\_len}}_{3.})$$

1. Data
2. Data length
3. Response area length

- An RPC request data file for the offline tester can also be used.

- An error occurs when the specified data is less than the specified data length. Data that exceeds the data length is ignored.

(2) XATMI request data file

An XATMI request data file stores the data passed to the service function for a requested service when using the client UAP simulator with an XATMI interface. A single file contains one set of data.

(a) File structure

Call type	Buffer type	Buffer subtype	Flags	Data length	Data

(b) File contents

Item	Position	Length (bytes)	Contents

Call type	0	8	Type of function calling a service:
			`call`
			call from `tpcall` function
			`acall`
			call from `tpacall` function
			`connect`
			call from `tpconnect` function

Buffer type	8	8	Buffer type, specified as one of the following character strings: • `X_OCTET` • `X_COMMON` • `X_C_TYPE`
Buffer subtype	16	16	Buffer subtype, specified as a string of up to 16 characters. Specify a null character when specifying `X_OCTET` as the buffer type.

Item	Position	Length (bytes)	Contents
Flags	32	4	Flags to be passed to the service function, specified as a hexadecimal and restricted by the specified call type: 0x00000000 0 (for `call` and `acall` only) 0x00000004 `TPNOREPLY` (for `acall` only) 0x00000008 `TPNOTRAN` 0x00000100 `TPNOCHANGE` (for `call` and `acall` only) 0x00000800 `TPSENDONLY` (for `connect` only) 0x00001000 `TPRECVONLY` (for `connect` only) `TPNOTIME` and `TPSIGRSTRT` are always set at service requests. `TPNOBLOCK` is not set.
Data length	36	4	Length of the data to be passed to the service function (0-524288). Specify zero when no data is passed. The buffer type and subtype specifications are ignored when zero is specified.
Data	40	*n*	Data to be passed to the service function

(c) Notes

- The items in the XATMI request data file are related to the service function arguments as follows:

```
void tpservice(svcinf)
     TPSVCINFO *svcinf;

     struct TPSVCINFO {
               char name[32];
               char *data;          1.
               long len;            2.
               long flags;
               int cd;
            }
```

1. Address at which the data mapped to the buffer type and subtype is stored
2. Length of the data shown by `data`

- The items in the XATMI request data file are related to the XATMI functions issued by the `utoxsppsvc` command as follows.

```
idata=tpalloc(type,subtype,ilen);
               2.     3.     4.
 tpcall(svc, idata,ilen,odata,olen,flags);
    1.             4.                5.
 tpacall(svc, idata,ilen,flags);
    1.              4.    5.
 tpconnect(svc, idata,ilen,flags);
    1.                4.    5.
```

1. XATMI function corresponding to the call type
2. Buffer type name
3. Buffer subtype name
4. Data length
5. Flags (specified as the actual flag values of the specified flags)

- An XATMI request data file for the offline tester can also be used.
- An error occurs when the specified data is less than the specified data length. Data that exceeds the data length is ignored.
- When the subtype name is less than 16 characters, add null characters to the end of the name.
- For buffer types other than `X_OCTET`, the data in the XATMI request data file is illegal when the subtype data length specified in the file differs from the subtype data length specified in the `utoxsppsvc` command.
- `TPNOCHANGE` can be specified in `flag`, but the specification is ignored.
- When the buffer type and subtype are specified, the values specified for the data length and data must be the same as the data structure value defined for the stubs.

  Boundary alignment is performed for the data structure defined for the stubs (the total length is an integer multiple of 4). For this reason, the user must consider the alignment portion when creating an XATMI request data file.

  Check boundary alignment details in the stub source created by the `stbmake` command and an output result created by the `stbmake` command with the `-p` option specified.

3.3.2 Service response data files

(1) RPC response data file

When using the server UAP simulator with an RPC interface, the RPC response data file stores the response data returned to the UAP making the service request to the simulate SPP. A single file contains one set of service data.

When using the client UAP simulator, the RPC response data file stores the response data returned from the test UAP.

(a) File structure

Data length	Data

(b) File contents

Item	Position	Length (bytes)	Contents
Data length	0	4	Length of the data to be returned to the UAP making the service request. (0-2147483647)
Data	4	*n*	Data to be returned to the UAP making the service request.

(c) Notes

- The items in the RPC response data file are related to the arguments of the service request function (`dc_rpc_call` function) of the UAP making the service request as follows:

```
dc_rpc_call(.....,in,in_len,out,out_len)
                            1.
```

1. Data

- An RPC response data file for the offline tester can also be used.
- An error occurs when the specified data is less than the specified data length. Data that exceeds the data length is ignored.

(2) XATMI response data file

When using the server UAP simulator with an XATMI interface, the XATMI response data file stores the response data returned to the UAP making the service request to the simulate SPP. A single file can contain one or more sets of data.

(a) File structure

Buffer type	Buffer subtype	Service termination code	Return code	Data length	Data
Buffer type	Buffer subtype	Service termination code	Return code	Data length	Data
:	:	:	:	:	:
Buffer type	Buffer subtype	Service termination code	Return code	Data length	Data

(b) File contents

Item	Position	Length (bytes)	Contents
Buffer type	0	8	Buffer type, specified as one of the following character strings: • `X_OCTET` • `X_COMMON` • `X_C_TYPE`
Buffer subtype	8	16	Buffer subtype, specified as a string of up to 16 characters. Specify a null character when specifying `X_OCTET` as the buffer type.

Service termination code	24	4	One of the following hexadecimal values of `rval` in the `tpreturn` function. The value is set in the `tperrno` area.
			0x04000000
			`TPSUCCESS`
			0x20000000
			`TPFAIL`

Return code	28	4	Hexadecimal value of `rcode` in the `tpreturn` function. The value is set in the `tpurcode` area.
Data length	32	4	Length of the data to be returned to the UAP making a service request (0-524288). Specify zero when no data is passed. The buffer type and subtype specifications are ignored when zero is specified.
Data	36	*n*	Data to be returned to the UAP making the service request.

(c) Notes

- The items in the XATMI response data file are related to the arguments of the service termination function (`tpreturn` function) as follows:

```
tpreturn(rval,rcode,data,len,.....)
          1.   2.    3.   4.
```

1. Service termination code
2. Return code
3. Data stored in the buffer allocated by buffer type and subtype
4. Data length

- An XATMI response data file for the offline tester can also be used.
- An error occurs when the specified data is less than the specified data length. Data that exceeds the data length is ignored.
- When the buffer type and subtype are specified, the values specified for the data

length and data must be the same as the data structure value defined for the stubs.

Boundary alignment is performed for the data structure defined for the stubs (the total length is an integer multiple of 4). For this reason, the user must consider the alignment portion when creating an XATMI response data file.

Check boundary alignment details in the stub source created by the `stbmake` command and an output result crated by the `stbmake` command with the `-p` option specified.

3.3.3 XATMI receive data file

An XATMI receive data file stores the messages received by the UAP in the `tprecv` function when making the conversational service paradigm. A single file can contain a number of data items which are passed consecutively to the `tprecv` function.

Create an XATMI receive data file for each service.

(1) File structure

Common area	Buffer type	Buffer subtype	Event flag	Data length	Data
Common area	Buffer type	Buffer subtype	Event flag	Data length	Data
: :	: :	: :	: :	: :	: :
Common area	Buffer type	Buffer subtype	Event flag	Data length	Data

(2) File contents

Item	Position	Length (bytes)	Contents
Common area	0	36	Area shared with the XATMI send data file. Specify a space or null character.
Buffer type	36	8	• Buffer type, specified as one of the following character strings: • `X_OCTET` • `X_COMMON` • `X_C_TYPE`
Buffer subtype	44	16	Buffer subtype, specified as a string of up to 16 characters. Specify a null character when specifying `X_OCTET` as the buffer type.

Item	Position	Length (bytes)	Contents

Event flag	60	4	One of the following hexadecimal values as the event flag to be passed to the `tprecv` function:
			0x00000000
			0
			0x00000001
			`TPEV_DISCONIMM`
			0x00000002
			`TPEV_SVCERR`
			0x00000004
			`TPEV_SVCFAIL`
			0x00000008
			`TPEV_SVCSUCC`
			0x00000020
			`TPEV_SENDONLY`

Data length	64	4	Length of the data to be passed to the `tprecv` function (0-524288). Specify zero when no data is passed. The buffer type and subtype specifications are ignored when zero is specified.
Data	68	*n*	Data to be passed to the `tprecv` function

(3) Notes

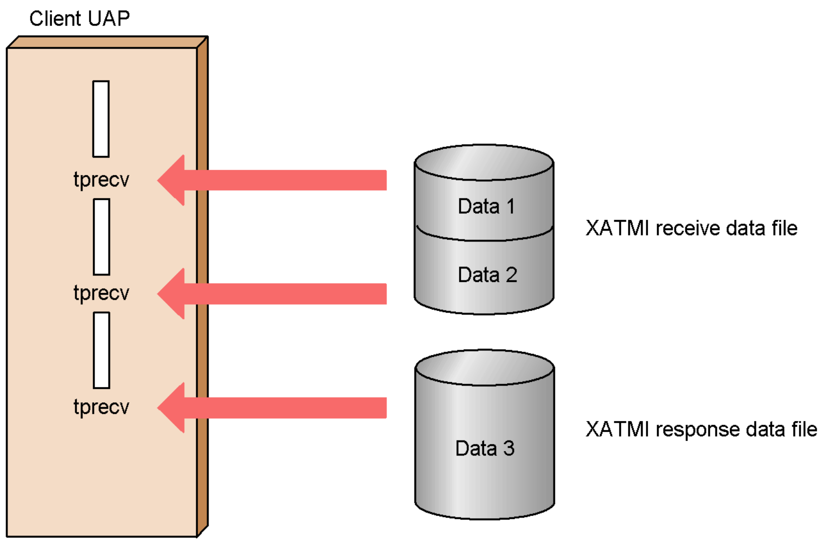
- The items in the XATMI receive data file are related to the arguments of the message receive function (`tprecv` function) as follows:

```
tprecv(.....,data,len,.....,revent)
              1.   2.         3.
```

1. Data stored in the buffer allocated by buffer type and subtype
2. Data length
3. Event flag

- Figure 3-1 shows the relationships between the data passed to the `tprecv` function and the XATMI receive data and XATMI response data files when using the server UAP simulator.

Figure 3-1: Receive data and tester files



- When using the server UAP simulator, create the receive data in execution units. If the `tprecv` function is issued more than once in a service, create all the data required for the number of executions. However, the data passed to the final `tprecv` function can be stored in an XATMI response data file.

  If the `tprecv` function is executed more times than the number of data items, the system assumes that data from the `tpreturn` function was received and an error occurs at each execution that exceeds the number of data items.

  The XATMI receive data file opens and closes by service unit.
- XATMI receive data files for the offline tester cannot be used. However, the `cat` command can be used to edit a number of XATMI receive data files into a single file for use with the online tester.
- An XATMI send data file containing the send data to be output when using the server UAP simulator can be used without modification as an XATMI receive data file.
- An error occurs when the specified data is less than the specified data length. Data that exceeds the data length is ignored.
- If a value other than `TPEV_SENDONLY` is specified as the event flag when using the server UAP simulator, the `tprecv` function issued by the client UAP receives events that cannot be continued interactively any further. Therefore, the remaining data items cannot be used. Zero is set in the global variable `tpurcode` when an event occurs.

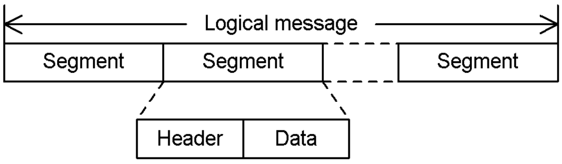
- `0` and `TPEV_SENDONLY` are the only valid specifications for the event flag when using the client UAP simulator. Other specifications are ignored.
- When the buffer type and subtype are specified, the values specified for the data length and data must be the same as the data structure value defined for the stubs.

  Boundary alignment is performed for the data structure defined for the stubs (the total length is an integer multiple of 4). For this reason, the user must consider the alignment portion when creating an XATMI receive data file.

  Check boundary alignment details in the stub source created by the `stbmake` command and an output result crated by the `stbmake` command with the `-p` option specified.

3.3.4 MCF receive message files

A logical message can contain one or more segments. A segment consists of a header part containing the segment information and a data part which is the message text.



There are five types of segments:

- Single segment

  Segment in a logical message consisting of one segment only
- First segment

  First segment in a logical message consisting of multiple segments
- Middle segment

  One of the middle segments in a logical message consisting of multiple segments
- Last segment

  Last segment in a logical message consisting of multiple segments
- Header segment

  Segment prefixed to two concatenated messages

Specify the segment type in the header part.

(1) Asynchronous receive message file

An asynchronous receive message file stores the messages received by the UAP in an MCF function (`dc_mcf_receive` function). Create one logical message per file.

When a header segment is used, the data is prefixed to the message.

(a) File structure

■ Logical message consisting of one segment only

Single segment	
Header	Data

■ Logical message consisting of multiple segments

First segment		Middle segment		Middle segment			Last segment	
Header	Data	Header	Data	Header	Data		Header	Data

■ Header segment

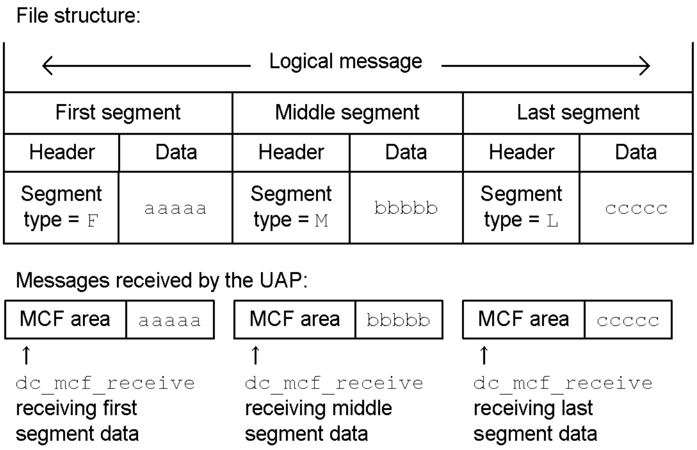
Header segment	
Header	Data

(b) File contents

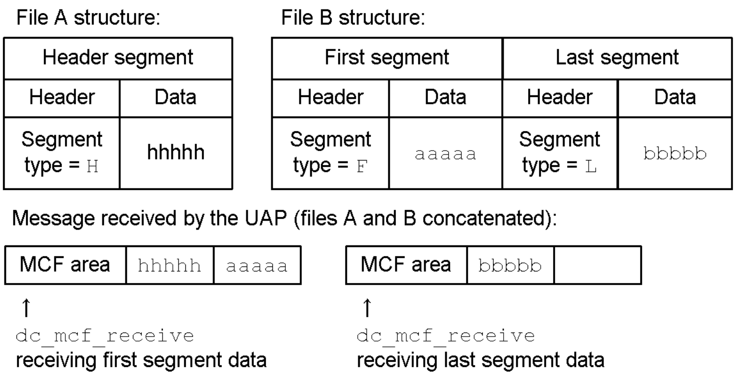
<table>
<tr><th colspan="2">Item</th><th>Position</th><th>Length (bytes)</th><th>Contents</th></tr>
<tr><td rowspan="5">Header</td><td>Input/output logical terminal name</td><td>0</td><td>9</td><td>Logical terminal name (including final null character) to be passed to MCF functions. Specify the same name for each segment of a multiple-segment message.</td></tr>
<tr><td>Map name</td><td>9</td><td>9</td><td>Map name (including final null character). Specify the same name for each segment of a multiple-segment message.
This specification is valid only for functions that return a map name.</td></tr>
<tr><td>Reserved</td><td>18</td><td>9</td><td>Null character</td></tr>
<tr><td>Segment type</td><td>27</td><td>1</td><td>One of the following characters:
<code>F</code>
First segment
<code>M</code>
Middle segment
<code>L</code>
Last segment
<code>O</code>
Single segment
<code>H</code>
Header segment</td></tr>
<tr><td>Message length</td><td>28</td><td>4</td><td>Message length (0-2147483647)</td></tr>
<tr><td>Data</td><td>Message</td><td>32</td><td>n</td><td>The data in the segment, of the specified message length</td></tr>
</table>

(c) Notes

- The following shows how the items in an asynchronous receive message file are related to message receive requests from a UAP via an MCF function.



- By concatenating header segments, data created in another file can be combined with the first or single segment and passed together to the UAP. The following shows how a header segment is related to a message receive request from a UAP by an MCF function.



- Segment types `F` (first segment) and `M` (middle segment) are handled in the same way. Also, segment types `L` (last segment) and `O` (single segment) are handled in the same way. For example, a file consisting of the three segment types `F`, `M`, and `L` is handled in the same way as a file consisting of segment types `M`, `M`, and `O`.
- The following shows the relationships between the segment type specified in the segment header for message send/receive with an MHP and the file type at execution. If the segment type is incorrectly specified, the receive request function returns an error at the first message receive.

Asynchronous receive message file containing segments other than header segments

When segment type L or O is specified for a message, the MHP regards the message as completed and ignores any subsequent segments.

Segment type			Segments received by MHP
First segment	Middle segment	Last segment	
F	M	L	F, M, L
F	L	X	F, L#1
L	X	X	L#2
X	M	L	No segments received.#3
F	X	L	

Legend:

X: Specification other than F, M, L, or O.

#1: At the third receive request, the MHP assumes that one logical message has been received and an error code is returned.

#2: The middle and subsequent segments are ignored.

#3: A message reports that the segment type is invalid and the receive request function returns an error code.

Asynchronous receive message file containing a header segment

Only the first segments in the file are valid.

Segment type	Segments received by MHP
H	H#1
H + X	
X	No segments received.#2
X + H	

Legend:

X: Specification other than H.

#1: However, the segment is passed in concatenated format with F, M, L, or O.

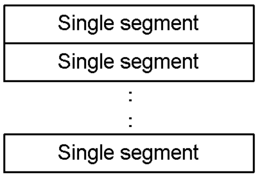
#2: A message reports that the segment type is invalid and the receive request function returns an error code.

(2) Synchronous receive message file

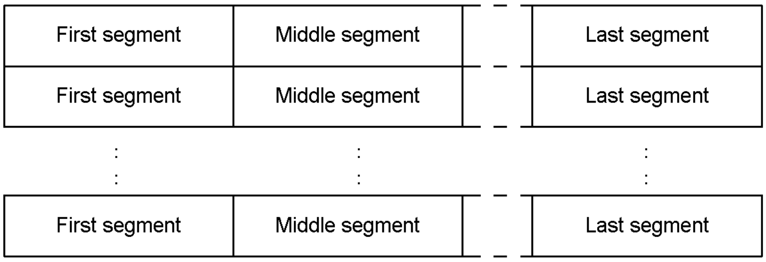
A synchronous receive message file stores the synchronous messages received by the UAP via MCF functions (`dc_mcf_recvsync` and `dc_mcf_sendrecv` functions). A single file can contain a number of logical messages. When a header segment is used, the data is prefixed to the message.

(a) File structure

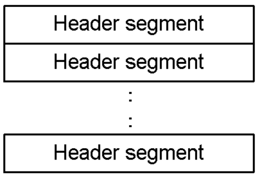
■ Logical message consisting of one segment only



■ Logical message consisting of multiple segments



■ Header segment



(b) File contents

Item		Position	Length (bytes)	Contents
Header	Input/output logical terminal name	0	9	Logical terminal name (including final null character) to be passed to MCF functions. Specify the same name for each segment of a multiple-segment logical message.
	Map name	9	9	Map name (including final null character). Specify the same name for each segment of a multiple-segment message. This specification is valid only for functions that return a map name.
	Reserved	18	9	Null character

	Segment type	27	1	One of the following characters:
				`F`
				First segment
				`M`
				Middle segment
				`L`
				Last segment
				`O`
				Single segment
				`H`
				Header segment

	Message length	28	4	Message length (0-2147483647)
Data	Message	32	*n*	The data in the segment, of the specified message length

(c) Notes

- The following shows how the items in a synchronous receive message file are related to message receive requests from a UAP by an MCF function.

<File structure>

First segment		Middle segments		Last segment	
Header	Data	Header	Data	Header	Data
Segment type=`F`	`aaaaa`	Segment type=`M`	`bbbbb`	Segment type=`L`	`ccccc`
First segment		Middle segments		Last segment	
Header	Data	Header	Data	Header	Data
Segment type=`F`	`ddddd`	Segment type=`M`	`eeeee`	Segment type=`L`	`fffff`

<Messages received by the UAP>

MCF area	`aaaaa`	← `dc_mcf_recvsync` function receiving first segment data
MCF area	`bbbbb`	← `dc_mcf_recvsync` function receiving middle segment data
MCF area	`ccccc`	← `dc_mcf_recvsync` function receiving last segment data

MCF area	`ddddd`	← `dc_mcf_recvsync` function receiving first segment data
MCF area	`eeeee`	← `dc_mcf_recvsync` function receiving middle segment data
MCF area	`fffff`	← `dc_mcf_recvsync` function receiving last segment data

- By concatenating header segments, data created in another file can be combined with the first or single segment and passed together to the UAP. The following shows how a header segment is related to a message receive request from a UAP by an MCF function.

<File A structure>

Header segment	
Header	Data
Segment type=H	h0001
Header segment	
Header	Data
Segment type=H	h0002

<File B structure>

First segment		Last segment	
Header	Data	Header	Data
Segment type=F	aaaaa	Segment type=L	bbbbb
First segment		Last segment	
Header	Data	Header	Data
Segment type=F	ccccc	Segment type=L	ddddd

<Messages received by the UAP (with files A and B concatenated)>

MCF area	h0001	aaaaa	← `dc_mcf_sendrecv` function receiving first segment data
MCF area	bbbbb		← `dc_mcf_recvsync` function receiving last segment data

MCF area	h0002	ccccc	← `dc_mcf_sendrecv` function receiving first segment data
MCF area	ddddd		← `dc_mcf_recvsync` function receiving last segment data

- When the MCF simulator is used and the UAP receives a number of logical messages synchronously, associate the header segment prefixed to each receive message with the appropriate logical message. If no header segment is required for any of the logical messages, set a dummy header segment, specifying 0 as the message length. If none of the logical messages require a header segment, there is no need to create a header segment file.

  The following shows the relationships between the header segment and the message receive requests from the UAP via MCF functions.

<File A structure>

Header segment	
Header	Data
Segment type=`H`	Null
Header segment	
Header	Data
Segment type=`H`	`h0001`

<File B structure>

First segment		Last segment	
Header	Data	Header	Data
Segment type=`F`	`aaaaa`	Segment type=`L`	`bbbbb`
First segment		Last segment	
Header	Data	Header	Data
Segment type=`F`	`ccccc`	Segment type=`L`	`ddddd`

<Messages received by the UAP (with files A and B concatenated)>

MCF area	`aaaaa`		← `dc_mcf_sendrecv` function receiving first segment data
MCF area	`bbbbb`		← `dc_mcf_recvsync` function receiving last segment data
MCF area	`h0001`	`ccccc`	← `dc_mcf_sendrecv` function receiving first segment data
MCF area	`ddddd`		← `dc_mcf_recvsync` function receiving last segment data

- Segment types `F` (first segment) and `M` (middle segment) are handled in the same way. Also, segment types `L` (last segment) and `O` (single segment) are handled in the same way. For example, a file consisting of the three segment types `F`, `M`, and `L` is handled in the same way as a file consisting of segment types `M`, `M`, and `O`.
- The following shows the relationships between the segment types specified in the segment headers for message send/receive with an MHP and the file types at execution. If a segment type is incorrectly specified, the receive request function returns an error at the first message receive.

■ Synchronous receive message file containing segments other than header segments

When segment type `L` or `O` is specified for a message, the MHP regards the message as completed and ignores any subsequent segments.

Segment type			Segments received by MHP
First segment	**Middle segment**	**Last segment**	
`F`	`M`	`L`	(`F`, `M`, `L`)
`M`	`M`	`L`	(`M`, `M`, `L`)
`O`	`O`	`O`	(`O`), (`O`), (`O`)
`F`	`L`	`M`	(`F`, `L`), (`M`)

Segment type			Segments received by MHP
First segment	Middle segment	Last segment	
X	M	L	No segments received.#
F	X	L	

Legend:

X: Specification other than F, M, L, or O.

( ): One logical message

#: A message reports that the segment type is invalid and the receive request function returns an error code.

■ Synchronous receive message file containing a header segment

All the header segments in the file are valid.

Segment type	Segments received by MHP
H	H#1
H + H	H, H#1
H + X	No segments received.#2
X	
X + H	

Legend:

X: Specification other than H.

#1: However, the segment is passed in concatenated format with F, M, L, or O.

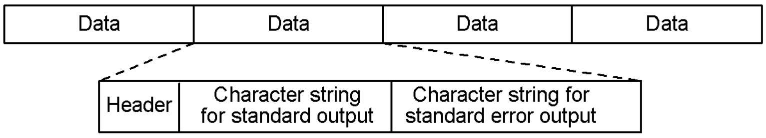
#2: A message reports that the segment type is invalid and the receive request function returns an error code.

3.3.5 Operating command result data file

An operating command result data file stores the data returned to the UAP as the command execution result when using the operating command simulator. A single file contains all the data required for the number of executions of the dc_adm_call_command function in one service.

Create an operating command result data file for each service.

(a) File structure



(b) File contents

Item		Position	Length (bytes)	Contents
Header	Operating command result code	0	4	Result code value set in the `stat` argument of the `dc_adm_call_command` function
	Character string length for standard output	4	4	Length of character strings (including null characters) output to standard output (0-2147483647)
	Character string length for standard error output	8	4	Length of character strings (including null characters) output to standard error output (0-2147483647)
Character string for standard output		12	*n*	Value set in the `outmsg` argument of the `dc_adm_call_command` function. (Includes the final null character. If no null characters are added, the last character is replaced with a null character.) The specified value is ignored when zero is specified as the character string length for standard output.
Character string for standard error output		--	*n*	Value set in the `errmsg` argument of the `dc_adm_call_command` function. (Includes the final null character. If no null characters are added, the last character is replaced with a null character.) The specified value is ignored when zero is specified as the character string length for standard error output.

Legend:

--: Not applicable

(c) Notes

- An operating command result data file for the offline tester can also be used. However, when the `dc_adm_call_command` function is issued more than once in a service, all the data (files) for the number of executions must be edited into a single file by the `cat` command.
- Add a null character to the end of the character strings for standard output and

standard error output. If no null character is specified, the last character in the string is replaced with a null character. If 0 is specified as the character string length, the specified string is ignored.

- When issuing operating commands by `SEND` statement in a DML, specify the data part as follows:

  Character string length for standard output:

    Specify 0.

  Character string length for standard error output:

    Specify 0 (when standard error output is not available).

3.4 Creating files

This section provides details about how the directory used for storing tester files is created, and how the user can create test data definition files to simplify later creation of tester files. This section also provides a list of the files that the online tester creates.

3.4.1 Test directory

The `$DCDIR/spool/uto` directory for storing tester files is created by OpenTP1 in mode 0777 at installation of the online tester.

Also, if no `$DCDIR/spool/uto/`*test-user-ID* directory exists at creation of a trace file or MCF send message file during UAP execution, the online tester creates the directory in mode 0777.

The user must create the `$DCDIR/spool/uto/`*test-user-ID* directory (or `$DCDIR/spool/uto/`*test-user-ID*`/`*user-server-name* directory if required) when creating a MCF send message file or other online tester file prior to testing. Set the mode to enable creation of the above files during UAP execution.

3.4.2 Test data definition file

By creating a *test data definition file*, the user can easily create tester files using the tester file creation facility.

A test data definition file can have any name. The following tester files can be created from a test data definition file:

- RPC request data file
- XATMI request data file
- RPC response data file
- XATMI response data file
- XATMI receive data file
- Asynchronous receive message file
- Synchronous receive message file
- Operating command result data file

To create a test data definition file:

1. Use a text editor to create a test data definition file.
2. Check the contents of the file and close the file.
3. Specify the created test data definition file in the `utofilcre` command and execute the command.

A tester file is created.

(1) Syntax

```
# comment            ...........1.
start tester-file-identifier tester-file-type output-file-name            ...........2.
keyword = input-data           ...........5.
keyword = input-data
sep         ...........3.
keyword = input-data
   :  :
   :  :
keyword = input-data
end         ...........4.
```

Note that the italicized numbers above correspond to the numbers under *(3) Explanation* below.

(2) Function

Enables tester files to be created by tester file creation command from the test data defined in the definition file.

One line in the definition file can be up to 512 bytes in length, including the line feed code.

(3) Explanation

1. Comment statement

   comment

   Write a one-line comment beginning with #.

2. `start` statement

   Declares the start of the input data for one tester file. Write a `start` statement before the input data for each tester file.

   When a test data definition file contains input data for two or more tester files, write an `end` statement at the end of input data in each tester file.

   - *tester-file-identifier* ~<up to 14 alphanumerics>

     Specify an identifier for each set of the tester file data created in the test data definition file. The identifiers must be unique within a definition file. Use alphanumerics `a-z`, `A-Z`, and `0-9` for an identifier.

   - *tester-file-kind*

     Specify the tester file kind as one of the following:

     `RRQ`

RPC request data file

`XRQ`

XATMI request data file

`RRT`

RPC response data file

`XRT`

XATMI response data file

`XRV`

XATMI receive data file

`NRV`

Asynchronous receive message file

`SRV`

Synchronous receive message file

`COM`

Operating command result data file

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

  Specify the name of the tester file to be created from the input data.

  When creating input data for two or more tester file kinds in one definition file, specify different output file names for each file kind.

  If the same output file name is specified for input data items for different tester file kinds, the test data is added to the specified file when the file is created. No error occurs, but the tester file created from the data may not be usable for a test.

  When an existing file name is specified, the test data is added to the specified file when the file is created.

3. `sep` statement

   Delimits input data items when a tester file is to contain multiple data items. `sep` statements can be specified when creating the following tester files:

   - XATMI receive data file
   - Synchronous receive message file
   - Operating command result data file

4. `end` statement

Declares the end of the input data for one tester file. Write an `end` statement after the input data for each tester file.

5. Input data definition statement

   Defines the input data for each tester file.

   Input data can consist of *fixed-information data* which can be set in advance and *user data* (`data` keyword) which can be any information set by the user. Write all the fixed-information data before the user data for a tester file.

   Input data cannot be duplicated within the test data for a tester file. The exception is an operating command result data file, for which user data must be specified twice (character string data for standard output and for standard error output).

   For details about the input data formats for specifying fixed information data, see the tables in *(5) Formats for the input data corresponding to the keywords of tester files*, below.

   - *keyword*

     Specify keywords to identify the data specific to each tester file. Space characters and tab codes before or after a keyword are ignored.

   - *input-data*

     Specify the input data for each keyword. Space characters and tab codes before or after the input data are ignored.

(4) Required settings for specifying user data as input data

The formats of user input data are described below.

(a) Setting user data length

Set the data length of user data as fixed-information data in the following format:

```
data_len=bytes
```

If the user data exceeds the value set in `data_len`, the message is truncated at output. If the user data is less than the value set in `data_len`, no further data can be set.

Example:

```
data_len=5
data='1234567'
```

→ Data:

31	32	33	34	35

```
data_len=5
data='123'
```

→ Data:

31	32	33	00	00

(b) Initializing user data

Use the tester file creation command to initialize the user data in the specified data length.

(c) Setting character data

Set character data in the following format:

```
data='data'
```

Do not add a null character to the end of character data.

Example:

`data='12345'` → Data: | 31 | 32 | 33 | 34 | 35 |

(d) Setting binary data

Set binary data in the following format:

```
data=data
```

Data can be written in decimal and hexadecimal notation, as follows:

- Decimal

  Set numeric values as is.

- Hexadecimal

  Prefix `0x` to numeric values.

Example:

`data=5` ➔ Data: 5 in decimal notation

`data=0x05` ➔ Data: 5 in hexadecimal notation

Binary data is set as the `int` datatype.

(e) Setting special characters

Line feed codes, tab codes, null characters, apostrophes ('), and the \ symbol are handled as special characters in character data. Specify these characters as follows:

Special character	Coding format
Line feed code	\n
Tab code	\t
Null character	\0

Special character	Coding format
'	\'
\	\\

(f) Loading user data from a file

To load user data from a file, set the data in the following format:

`data=(file)` *file-pathname*

Example:

`data=(file)/tmp/datafile` ➜ Data in `/tmp/datafile` is set.

(g) Setting the starting position of user data

User data can be set from any position, using the following format:

`data=[`*offset-from-start-of-user-data*`]` *data*

Example:

```
data_len=10
data=[2]'1234'
```

→ Data:

00	00	31	32	33	34

(h) Setting multiple data types

When using two or more data types, set the user data in the following format:

`data=`*data*
`=`*data*
:
:

Example:

`data=0x00000001` ➜ Data: First

`='ABCDEF'` ➜ Data: Second

(i) Aligning boundaries

When different data types are specified, the tester file creation command automatically sets the second data at the boundary of the first data. However, boundary alignment is not performed when:

- User data is loaded from a file
- The starting position of the user data is set

Example:

```
data_len=20
data='12345'
    =10
    =[15]'6789'
```

Boundaries aligned

→ Data:

31	32	33	34	35	00	00	00	00	00
00	0a	00	00	00	36	37	38	39	00

(5) Formats for the input data corresponding to the keywords of tester files

The following tables list the keywords and formats of the corresponding input data for each tester file.

For details about the type of information to be specified, see the description of each tester file in Section *3.3 User-created files*.

Table 3-5: Keywords and input data formats for RPC request data files

Keyword	Information	Explanation
`out_len`	Response area length	Length of the response area for the `dc_rpc_call` function. Specify a decimal or hexadecimal. Set before `data`.
`data_len`	Data length	Length of the user data to be passed to the server UAP by the `dc_rpc_call` function. Specify a decimal or hexadecimal. Set before `data`.
`data`	Data	User data to be passed to the server UAP by the `dc_rpc_call` function.

Table 3-6: Keywords and input data formats for XATMI request data files

Keyword	Information	Explanation
`call_kind`	Call type	Type of service request function. Set one of the following character strings: • `call` • `acall` • `connect` Set before `data`.
`buff_type`	Buffer type	Set one of the following character strings: • `X_OCTET` • `X_COMMON` • `X_C_TYPE` Set before `data`.
`sub_type`	Buffer subtype	Specify a string of up to 16 characters. *Example:* `sub_type=subtype01` Set before `data`.

Keyword	Information	Explanation
`flag`	Flags	One or more flags to be passed to the service function. Set any of the following character strings and delimit with vertical lines (|): • `0` • `TPNOREPLY` • `TPNOTRAN` • `TPNOCHANGE` • `TPSENDONLY` • `TPRECVONLY` Set before `data`.
`data_len`	Data length	Length of the user data to be passed to the server UAP by the `tpcall`, `tpacall`, or `tpconnect` function. Specify a decimal or hexadecimal. Set before `data`.
`data`	Data	User data to be passed to the server UAP by the `tpcall`, `tpacall`, or `tpconnect` function.

Table 3-7: Keywords and input data formats for RPC response data files

Keyword	Information	Explanation
`data_len`	Data length	Length of the user data to be returned to the client UAP at completion of a service. Specify a decimal or hexadecimal. Set before `data`.
`data`	Data	User data to be returned to the client UAP at completion of a service.

Table 3-8: Keywords and input data formats for XATMI response data files

Keyword	Information	Explanation
`buff_type`	Buffer type	Set one of the following character strings: • `X_OCTET` • `X_COMMON` • `X_C_TYPE` Set before `data`.
`sub_type`	Buffer subtype	Specify a string of up to 16 characters. *Example:* `sub_type=subtype01` Set before `data`.
`rval`	Service termination code	Specify one of the following character strings: • `TPSUCCESS` • `TPFAIL` Set before `data`.
`rcode`	Return code	Specify a decimal or hexadecimal. Set before `data`.

Keyword	Information	Explanation
data_len	Data length	Length of the user data to be returned to the client UAP at completion of a service. Specify a decimal or hexadecimal. Set before data.
data	Data	User data to be returned to the client UAP at completion of a service.

Table 3-9: Keywords and input data formats for XATMI receive data files

Keyword	Information	Explanation
buff_type	Buffer type	Set one of the following character strings: • X_OCTET • X_COMMON • X_C_TYPE Set before data.
sub_type	Buffer subtype	Specify a string of up to 16 characters. *Example:* sub_type=subtype01 Set before data.
event	Event flag	Event flag to be passed to the tprecv function. Specify one of the following character strings: • 0 • TPEV_DISCONIMM • TPEV_SVCERR • TPEV_SVCFAIL • TPEV_SVCSUCC • TPEV_SENDONLY Set before data.
data_len	Data length	Length of the user data to be passed to the tprecv function. Specify a decimal or hexadecimal. Set before data.
data	Data	User data to be passed to the tprecv function.
sep	sep statement	Write at the end of the data for one service when coding data for a number of services. Do not set at the end of the final data.

Note

When coding data for a number of services, repeat the data specifications from buff_type onwards.

Table 3-10: Keywords and input data formats for asynchronous receive message files

Keyword	Information	Explanation
`termname`	Input/output logical terminal name	Name of the I/O logical terminal to be passed to the `dc_mcf_receive` function. Specify a string of up to 8 characters. Set before `data`.
`mapname`	Map name	Map name to be passed to the `dc_mcf_receive` function. Specify a string of up to 8 characters. Set before `data`.
`seg_kind`	Segment type	Segment type to be passed to the `dc_mcf_receive` function. Specify one of the following characters: • `F` • `M` • `L` • `O` • `H` To set data for multiple segments, use any of the following sequences: • `F...M...L` • `F...F...L` • `M...M...L` • `L` • `H` • `O` Set before `data`.
`data_len`	Message length	Length of the user data in the segment to be passed to the `dc_mcf_receive` function. Specify a decimal or hexadecimal. Set before `data`.
`data`	Message	User data in the segment to be passed to the `dc_mcf_receive` function.

Note

When setting data for a number of segments, repeat the data specifications from `seg_kind` onwards.

Table 3-11: Keywords and input data formats for synchronous receive message files

Keyword	Information	Explanation
`termname`	Input/output logical terminal name	Name of the I/O logical terminal to be passed to the `dc_mcf_recvsync` and `dc_mcf_sendrecv` functions. Specify a string of up to 8 characters. Set before `data`.
`mapname`	Map name	Map name to be passed to the `dc_mcf_recvsync` and `dc_mcf_sendrecv` functions. Specify a string of up to 8 characters. Set before `data`.

Keyword	Information	Explanation
`seg_kind`	Segment type	Segment type to be passed to the `dc_mcf_recvsync` and `dc_mcf_sendrecv` functions.Specify one of the following characters: • `F` • `M` • `L` • `O` • `H` To set data for multiple segments, use any of the following sequences: • `F`...`M`...`L` • `F`...`F`...`L` • `M`...`M`...`L` • `F`...`M`...`L`...`F`...`M`...`L` • `F`...`F`...`L`...`M`...`M`...`L` • `M`...`M`...`L`...`M`...`M`...`L` • `L` • `H` • `O` • `L`...`L`...`F`...`M`...`L`...`L` • `O`...`O`...`F`...`M`...`L`...`O` • `H`...`H`...`H` Set before `data`.
`data_len`	Message length	Length of the user data in the segment to be passed to the `dc_mcf_recvsync` or `dc_mcf_sendrecv` function. Specify a decimal or hexadecimal. Set before `data`.
`data`	Message	User data in the segment to be passed to the `dc_mcf_recvsync` or `dc_mcf_sendrecv` function.
`sep`	`sep` statement	Write at the end of the data for one message when coding data for a number of messages. Do not set at the end of the final data.

Notes

1. When setting data for a number of messages, repeat the data specifications from `termname` onwards.
2. When setting data for a number of segments, repeat the data specifications from `seg_kind` to `data`.

Table 3-12: Keywords and input data formats for operating command result data file

Keyword	Information	Explanation
`status_code`	Operating command result code	Specify a result code returned from the operating command in decimal or hexadecimal. Set before `data`.
`outsize`	Message length for standard output	Length of the message output by operating command to standard output. Specify a decimal or hexadecimal. Set before `data`.
`errsize`	Message length for standard error output	Length of the message output by operating command to standard error output. Specify a decimal or hexadecimal. Set before `data`.
`data`	Character string for standard output	Message output by operating command to standard output. Set character data.
`data`	Character string for standard error output	Message output by operating command to standard error output. Set character data.
`sep`	`sep` statement	Write at the end of the data for one command when coding data for a number of commands. Do not set at the end of the final data.

Note

When coding data for a number of commands, repeat the data keywords and items from `status_code` onwards.

3.4.3 Files created by the online tester

The following tables list the types and names of files that the online tester creates when it is used.

Table 3-13: List of files created by online tester

File type		Use and contents	Time of creation	Deleted by	Time of deletion
Service response data files	RPC response data file	Stores data returned as the service result when using the client UAP simulator with an RPC interface.	At return of the service request#1	User	Any
	XATMI response data file	Stores data returned as the service result when using the client UAP simulator with an XATMI interface.	At return of the service request#1	User	Any

File type	Use and contents	Time of creation	Deleted by	Time of deletion
XATMI send data file	Stores data sent by the `tpsend` function when using a UAP simulator for making interactive service requests with an XATMI interface.	In the `tpsend` function#2	User	Any
MCF send message file	Stores messages send by the following functions when using the MCF simulator: • `dc_mcf_reply` • `dc_mcf_send` • `dc_mcf_sendsync` • `dc_mcf_sendrecv` • `dc_mcf_execap`	In the functions listed at left#2	User	Any
Temporary memory data file	Stores data updated by the `dc_mcf_tempput` function and acquired by the `dc_mcf_tempget` function in the UAP when using the MCF simulator.	In the `dc_mcf_tempput` and `dc_mcf_tempget` functions#1	Online tester#3	At execution of the `dc_mcf_contend` function
Trace file	Collects UAP trace information for an OpenTP1 function.	When the online tester (UAP) collects the first trace information.	User	Any#4

#1: If the file already exists, the existing data is overwritten by the new input data.

#2: If the file already exists, the new input data is added to the file.

#3: When not running a UAP that issues the `dc_mcf_contend` function, the user can delete the file at any time.

#4: The user can delete the file when full after backup to another file.

Table 3-14: Names for tester files created by the online tester

Tester file type		File name
Service response data files	RPC response data file	File name specified by the `utosppsvc` command
	XATMI response data file	File name specified by the `utoxsppsvc` command
XATMI send data file		`$DCDIR/spool/uto/`*test-user-ID*`/`*user-server-name*`/`*xsd-service-name*#

Tester file type		File name
MCF send message file		`$DCDIR/spool/uto/`*test-user-ID*`/sendmsg`
Temporary memory data file		`$DCDIR/spool/uto/`*test-user-ID*`/` *utotmp-logical-terminal-name*
Trace files	File 1	`$DCDIR/spool/uto/`*test-user-ID*`/trace1`
	File 2	`$DCDIR/spool/uto/`*test-user-ID*`/trace2`

#: When the service name exceeds 11 characters, the first five and last six characters are combined as the service name.

Example: Service name `uapservice0001` ➜ `uapsece0001`

When the service name exceeds 15 characters, the first five and the 10th to 15th characters are combined as the service name.

Example: Service name `uapxatmiservice0001` ➜ `uapxaervice`

Chapter

4. Test Execution

This chapter explains how to run a test with the online tester.

This chapter contains the following sections:

4.1 Creating UAPs
4.2 Service requests to an SPP
4.3 Service requests to an MHP
4.4 Creating tester files
4.5 Editing test information

4.1 Creating UAPs

To create a UAP that does not use the MCF simulator, follow the same procedure as for a job UAP. See the manual *OpenTP1 Programming Guide* for details.

To create a UAP that uses the MCF simulator, use the simulation functions library provided by the online tester. The creation procedure differs depending on whether TP1/Message Control is cataloged in the Resource Manager.

If TP1/Message Control is not cataloged in the Resource Manager, link the UAP to the online tester's MCF simulation functions library (`libmuto.a`) rather than to the TP1/Message Control library (`libmcf.a`).

Specify `-lmuto` to link the UAP to the MCF simulation functions library. There is no need to specify `-lmcf` to link the UAP to the TP1/Message Control library.

For a UAP created in COBOL or in a data manipulation language (DML), specify `-lmuto` instead of `-lmcf` in the same way.

If TP1/Message Control is cataloged in the Resource Manager, link the UAP first to the MCF simulation functions library (`libmuto.a`) and then to the TP1/Message Control library (`libmcf.a`).

The command for compiling a UAP that uses the MCF simulator is shown below.

■ TP1/Message Control not cataloged in the Resource Manager

```
cc  -go example exmain.c exsv1.c exsv2.c ex_sstb.c
    -l$DCDIR/include -L$DCDIR/lib -Wl, -B,immediate -Wl,
    -a,default -lmuto -lbetran -L/usr/lib -ltactk -lbsd -lc
```

Legend:

`exmain.c`: Main function

`exsv1.c`: Service function 1

`exsv2.c`: Service function 2

`ex_sstb.c`: Stub source created by the stub

■ TP1/Message Control cataloged in the Resource Manager

```
cc  -go example exmain.c exsv1.c exsv2.c ex_sstb.c
    -l$DCDIR/include -L$DCDIR/lib -Wl, -B,immediate -Wl,
    -a,default -lmuto -lmcf -lbetran -L/usr/lib -ltactk -lbsd -lc
```

Legend:

`exmain.c`

Main function

`exsv1.c`

Service function 1

`exsv2.c`

Service function 2

`ex_sstb.c`

Stub source created by the stub

4.2 Service requests to an SPP

This section describes how service requests are issued to an SPP when a client UAP or a server UAP is being simulated.

4.2.1 Client UAP simulator

(1) Simulating a client UAP with an RPC interface

Execute the `utosppsvc` command to simulate a client UAP that uses an RPC interface. Service requests can be sent to the SPP by issuing the `dc_rpc_call` function during command processing.

(2) Simulating a client UAP with an XATMI interface

Execute the `utoxsppsvc` command to simulate a client UAP that uses an XATMI interface. Service requests can be sent to the SPP by issuing the following functions during command processing:

- `tpcall` or `tpacall` function for the request/response service paradigm
- `tpconnect` function for the conversational service paradigm

4.2.2 Server UAP simulator

(1) Simulating a server UAP with an RPC interface

To simulate a server UAP that uses an RPC interface, activate the SPP (to which service requests are sent) as a dummy SPP. Specify `dmyspp` in the `test_mode` operand of the user service definition to create the dummy SPP.

To activate the dummy SPP, enter the OpenTP1 `dcsvstart` command. To send a service request to the dummy SPP, issue the `dc_rpc_call` function.

Execute the OpenTP1 `dcsvstop` or `dcstop` command to terminate the dummy SPP.

(2) Simulating a server UAP with an XATMI interface

To simulate a server UAP that uses an XATMI interface, activate the SPP (to which service requests are sent) as a dummy SPP. Specify `dmyspp` in the `test_mode` operand of the user service definition to create the dummy SPP.

Execute the OpenTP1 `dcsvstart` command to activate the dummy SPP and the `dcsvstop` or `dcstop` command to terminate the dummy SPP.

When the conversational service paradigm is sent to the server UAP simulator, the table that manages conversational status remains in the tester daemon if the process or service in the client UAP terminates without receiving an event flag indicating service completion in the `tprecv` function. In this case, terminate and then restart the dummy SPP.

4.3 Service requests to an MHP

To use the MCF simulator, activate the test MHP as a simulate MHP. Specify `simmhp` in the `test_mode` operand of the user service definition to create the simulate MHP.

To activate the simulate MHP, execute the OpenTP1 `dcsvstart` command or specify `dcsvstart` in the user service configuration definition.

To send a service request to the MHP, enter the `utomhpsvc` command. If a service request is sent in any other way, the online tester outputs an error message and skips execution of the requested service. In this case, the `dc_rpc_call` function terminates normally because the online tester accepts the service request, but response data for the service is not guaranteed.

The simulate MHP is activated as an SPP. This means that SPP commands must be used to run the simulate MHP. However, the `utosppsvc` command cannot be used.

To terminate the simulate MHP, execute the OpenTP1 `dcsvstop` command or `dcstop` command.

4.4 Creating tester files

Enter the `utofilcre` command to create a tester file.

In tester file creation, how to create a tester file or enter a command depends on whether to use the test data definition file or to use data output from the operating command.

4.4.1 Creating tester files using the test data definition file

The following shows how to create tester files using the test data definition file.

Example:

To create an RPC response data file and an operating command result data file:

1. Open the test data definition file using a text editor.

'vi testenv_file'

2. Set the input data for the RPC response data file and operating command result data file.

vi editor: Contents of `testenv_file`

```
# Data definition 1 for RPC response data file
# start test1 RRT /tmp/rpcrtnf01
data_len=20
data='abcdefg'
    =0x0008
end
# Data definition 2 for RPC response data file
# start test2 RRT /tmp/rpcrtnf02
data_len=20
data='abcdefg'
    =0x0008
end
# Data definition for operating command result
# data file start test3 COM /tmp/comrtnf03
status-code=-1
outsize=20
errsize=10
data='abcdefg'
data='abcdefg'
end
```

3. Check the coding, then close the test data definition file.
4. Execute the `utofilcre` command, specifying the test data definition file.

'utofilcre -e testenv_file'

4.4.2 Creating tester files using operating command output data

The following shows how to create tester files using operating command output data.

Example:

To create an RPC request data file:

1. Determine trace data used as test data for editing and outputting an RPC trace data file. In this example, use data with trace number 6.

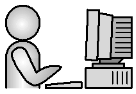
'rpcdump rpctrc_file'

2. Output the intended RPC trace data in the trace data file format to create a file.

'rpcdump -r -n6,6 rpctrc_file > testdata_file'

3. Execute the `utofilcre` command by specifying the tester file name, tester file kind, and a file that contains the RPC trace data.

'utofilcre -o rpcreqfile -k RRQ -i testdata_file'



4.5 Editing test information

4.5.1 Displaying test status

Execute the utols command to display test status when using the online tester. The following information can be displayed:

- Test mode of the UAP (value specified in the test_mode operand of the user service definition)
- Test user ID for the user who started the UAP
- Server name
- Service group name

See Section *5.1 Operating commands for running tests* in this part of the manual for the contents displayed.

4.5.2 Collecting UAP trace information

The online tester collects the same UAP trace information as OpenTP1. However, trace information specific to the online tester (tester information) can also be collected at the entrance to each OpenTP1 function.

To collect tester information, perform one of the following:

- Specify target or simmhp in the test_mode operand of the user service definition, or specify usable and activate the UAP in test mode.
- Specify 1 or a higher value in the uap_trace_max operand of the user service definition (or omit specification).
- Activate the UAP for which traces are to be collected by executing the OpenTP1 dcsvstart command or by specifying dcsvstart in the user service configuration definition.

Trace information is grouped by the online tester and output to a trace file at the times shown below. Tester information is output once only when the trace information is output to the trace file.

- At the start of the dc_rpc_mainloop function
- At the start of the dc_mcf_mainloop function
- At the start of the dc_rpc_call function
- At completion of the dc_rpc_close function
- At completion of an RPC service function
- At the start of the tpcall function

- At the start of the `tpacall` function
- At the start of the `tpconnect` function
- At completion of an XATMI service function

When the information for a group fills the UAP trace area, the information is output to the trace file. The UAP trace area is then reused from the beginning.

A trace file is created for each OpenTP1 system and for each test user ID. Therefore, if a number of UAPs that output trace information are executed in parallel, the UAP trace information is mixed and difficult to check. Parallel execution also results in waiting for release of locks and a timeout condition may occur before a reply can be made to a service request. For these reasons, parallel execution should be avoided when using the online tester.

A swap message is output when the size of one of the two trace files exceeds the value specified in the `max_trace_file_size` operand of the tester service definition. UAP traces are then collected in the other trace file. When both trace files are full, no further UAP trace information can be collected.

To prevent this situation, the user must copy the full trace file to another file when the swap message is output, and then delete the full trace file. If the second trace file subsequently becomes full, create a new file, specifying the name of the deleted trace file, and continue collecting trace information.

Do not delete a trace file while traces are still being collected. Deletion during trace collection means that no further information can be collected.

Note also that trace information for the `dc_trn_info` function is not collected.

4.5.3 Merging and outputting UAP trace information

To merge UAP trace information, execute the `utotrcmrg` command.

At input of the `utotrcmrg` command, the data in the specified trace files is ordered by service execution sequence in each group and is output to the specified file (trace merge file). If the specified output file already exists, its contents are deleted before the new merged data is written to the file.

Trace merge files have a different file type but the same format as trace files. Therefore, trace merge files can also be merged. Also, trace files can be merged during trace collection.

To edit and output UAP trace information, execute the `utotrcout` command. At command input, the data in the specified trace file or trace merge file is edited and output to standard output.

The following trace information can be output by executing the `utotrcout` command:

- Trace information for specific services
- Trace information for a specific server
- Trace information on function names and other selected items
- Trace information ordered in actual collection sequence
- Trace information collected within a specified time frame

See Section *5.1 Operating commands for running tests* in this part of the manual for details on output formats.

4.5.4 UAP traces for MCF simulation functions

UAP trace information is also collected for the MCF simulation functions. However, when information that the online tester cannot analyze is required (such as an MCF application definition), the trace information cannot be collected. In such cases, the online tester sets a dummy value.

Table 4-1 lists the dummy values set for trace information that the online tester cannot collect.

Table 4-1: Dummy values and non-collectable trace information

Function name	Non-collectable information	Dummy value
`dc_mcf_mainloop` (at start)	Application name	********
	Name of logical terminal where input	********
	Application type	0
`dc_mcf_mainloop` (at return)	Application name	********

The MCF simulation function `dc_mcf_mainloop` uses the `dc_rpc_mainloop` function. Therefore, trace information for `dc_rpc_mainloop` is also output when the `dc_mcf_mainloop` function is issued.

4.5.5 Editing and outputting send messages

To edit the send messages collected when using the MCF simulator, execute the `utomsgout` command. At command input, the data in the specified MCF send message file is edited and output to standard output or to a specified file.

The following trace information can be output by executing the `utomsgout` command:

- A list of abbreviated send messages
- Messages output in MCF receive message file format
- Messages not output in the MCF receive message file format

- Oldest send message
- Most recent send message
- Messages collected for a specific function
- Selected messages from a send message file
- Messages sent in a specific service

See Section *5.1 Operating commands for running tests* in this part of the manual for details on output formats.

4.5.6 Checking UAP response data

The data in the RPC and XATMI response data files output by the client UAP simulator can be output as an edited dump so that the file contents can be verified.

See Section *3.3 User-created files* in this part of the manual for details on the formats of the output data.

4.5.7 Checking UAP send data

The data in the XATMI send data files output by the server UAP simulator can be output as an edited dump so that the file contents can be verified.

See Section *3.3 User-created files* in this part of the manual for details on the formats of the output data.

The contents of the common area are as follows:

Item		Position	Length (bytes)	Contents
Common area	Service name	0	32	Stores the service names at the send destinations.
	Call descriptors	32	4	Stores the call descriptors used when sending service requests.

Chapter

5. Operating Commands

This chapter explains how to use the operating commands of the online tester.

This chapter contains the following section:

5.1 Operating commands for running tests

5.1 Operating commands for running tests

The following pages explain the online tester's operating commands. For information on command Syntax and rules, see the manual *OpenTP1 Operation*.

Table 5-1 lists the operating commands for running tests.

Table 5-1: List of operating commands

Command name	Function
`utodbgstop`	Termination of a UAP interlocked with the debugger
`utodebug`	Activation of a UAP interlocked with the debugger
`utofilcre`	Tester file creation
`utofilout`	Edited output of the tester file content
`utols`	Test status display
`utomhpsvc`	Service requests to an MHP
`utomsgout`	Edited output of send messages
`utosppsvc`	Service requests to an RPC interface SPP
`utotrcmrg`	Merger of UAP trace information
`utotrcout`	Edited output of UAP trace information
`utoxsppsvc`	Service requests to an XATMI interface SPP

5.1.1 utodbgstop (termination of a UAP interlocked with the debugger)

(1) Syntax

`utodbgstop [-f]` *server-name*

(2) Function

Requests to terminate a UAP that interlocks the debugger.

Execute the `utodbgstop` command in a window except one that was used to execute the `utodebug` command on the machine where the OpenTP1 system is operating.

After terminating the UAP using the `utodbgstop` command, also terminate the debugger as soon as possible. Until the debugger terminates, the `utodbgstop` or `utodebug` command remains in a response wait state.

When the `utodbgstop` command terminates the UAP, this UAP cannot restart with a debugger command.

If the specified server does not interlock the debugger, the command fails. The `utodbgstop` command is available only when the tester service is active.

(3) Option

- `-f`

  Forcibly terminate the specified server. When this specification is omitted, the corresponding server terminates normally.

(4) Command arguments

- *server-name* ~<identifier of 1-8 characters>

  Specify the name of the server corresponding to the debugger-interlocked UAP to be terminated.

(5) Notes

- Entering the command may issue the following message and condition codes, which can be ignored.

  Message ID

  KFCA01844-E

  Reason Code
  `STATUS`
  `EXIT`
  `ABORTING`
  `ABORT`

- When entering the command issues the following message and condition code, be sure to stop the debugger. No other actions are needed.

  Message ID

  KFCA01844-E

  Reason Code
  `CRITICAL`

5.1.2 utodebug (activation of a UAP interlocked with the debugger)

(1) Syntax

```
utodebug server-name
```

(2) Function

Requests to activate a debugger-interlocked UAP and identifies the window used to execute the `utodebug` command as an I/O interface with the debugger.

When executing the `utodebug` command, add `$DCDIR/bin`, `/usr/bin`, and `/bin` to the search path name when specifying the `prcsvpath` operand for the process service definition or the `prcpath` command.

Execute the `utodebug` command in a window on the machine where the OpenTP1 system is operating. One window allows to test one UAP interlocked with the debugger. The other commands are unexecutable in this window until the debugger terminates.

The command fails if neither `target` nor `simmhp` is specified for the `test_mode` operand in the user service definition on the specified server. The command also fails if the specified server is already active.

The `utodebug` command is available only when the tester service is active.

(3) Command arguments

■ *server-name* ~<identifier of 1-8 characters>

Specify the name of the server corresponding to the UAP to be tested by interlocking the debugger.

(4) Notes

- When the debugger-interlocked UAP terminates normally or abnormally, be sure to terminate the debugger.
- If a debugger process is terminated forcibly with the debugger interlocked, the debugger-interlocked UAP process may terminate incompletely, leaving part of the process unprocessed. Terminate the remaining process using the command.
- If the `utodebug` command is terminated forcibly while the debugger is interlocked, I/O operations for the debugger process coexist with I/O operations for the shell, disabling debugger control. To solve this, forcibly terminate the debugger process and the debugger-interlocked UAP process.
- If the debugger becomes uncontrollable during a test interlocked to the debugger, forcibly terminate the `utodebug` command process, the debugger process, and the UAP process interlocked to the debugger. If necessary, reexecute the `utodebug` command. Executing the `prcls` command shows the ID of the UAP process interlocked to the debugger process.

5.1.3 utofilcre (tester file creation)

(1) Syntax

```
utofilcre{-e test-data-definition-file-name|
          -o tester-file-name|-k tester-file-kind
          [-i input-data-file-name]}
```

(2) Function

Creates a tester file using the specified test data definition file or record data from the unload journal file or RPC trace data retrieved by the operating command.

(3) Options

- `-e` *test-data-definition-file-name* ~<pathname>

  Specify the test data definition file that defines input data for a tester file to be created.

  This option cannot be specified concurrently with the `-o`, `-k`, or `-i` option.

- `-o` *tester-file-name* ~<pathname>

  Specify the name of a tester file consisting of data that is extracted by the operating command. When specifying this option, also specify the `-k` option.

  The `-o` option cannot be specified concurrently with the `-e` option.

- `-k` *tester-file-kind*

  Specify the kind of a tester file consisting of data that is extracted by the operating command. Specifiable file kinds are:

  `RRQ`

  RPC request data file

  `RRT`

  RPC response data file

  `XRQ`

  XATMI request data file

  `XRT`

  XATMI response data file

  `XRV`

  XATMI receive data file

  `NRV`

Asynchronous receive message file

`SRV`

Synchronous receive message file

The operating command result data file cannot be made of data extracted by a command. Accordingly the `-k` option cannot specify the operating command result data file.

When specifying this option, also specify the `-o` option.

The `-k` option cannot be specified concurrently with the `-e` option.

■ `-i` *input-data-file-name* ~<pathname>

Specify the name of an input data file that stores data extracted by the operating command. When specifying this option, also specify the `-o` option.

When the `-o` option is specified and the `-i` option is omitted, the standard input is assumed.

The `-i` option cannot be specified concurrently with the `-e` option.

(4) Notes

- When the `-o` option is specified and the `-i` option is omitted, the standard input is assumed. This time, specify an input file using a pipe or redirection. When no input file is specified, the command waits for an input. To solve this, forcibly terminate the command.
- No map name is contained in mj record data of the unload journal file. When the `-o` option is specified to create an asynchronous receive message file or synchronous receive message file, specifying `mj` record data as input data assumes `UTOMAP` to be a map name by default.

5.1.4 utofilout (edited output of the tester file content)

(1) Syntax

```
utofilout -k tester-file-kind tester-file-name
```

(2) Function

Edits the contents of the specified tester file in a data format of the specified tester file kind and outputs the edited file to the standard output.

The tester file kind must be of a tester file to be edited and output. If a different tester file kind is specified, its data format is used for editing data. If the data is editable, the edited result is output. If the data cannot be edited, the command fails.

(3) Option

- `-k` *tester-file-kind*

  Specify the kind of a tester file to be edited and output. Specifiable tester file kinds are:

  `RRQ`

  RPC request data file

  `RRT`

  RPC response data file

  `XRQ`

  XATMI request data file

  `XRT`

  XATMI response data file

  `XRV`

  XATMI receive data file and XATMI send data file

  `NRV`

  MCF receive message file (asynchronous receive message file and synchronous receive message file)

  `COM`

  Operating command result data file

(4) Command arguments

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

  Specify the name of the tester file to be edited.

(5) Output format (-k option = RRQ)

```
file kind=RPC request data file (RRQ)                               ] 1.
file name=/tmp/rrqfile                                              ]
----------------------------------------------------------------
No.1                                                                ] 2.
  response area size=256                                            ] 3.
  data length=260                                                   ]
  data contents
00000000   52504320 72657175 65737420 64617461 RPC request data     ] 4.
00000010   00000000 00000000 00000000 00000000 ..............       ]
00000020 - 000000ff :     SAME DATA                                 ]
00000100   00000000                                                 ]
   5.                      6.                         7.
```

Legend:

1. File information
2. Data number
3. Specific information data
4. User data

   The same data is displayed as follows.

   (First matched data location) - (last matched data location) `: SAME DATA`
5. User data location
6. Hexadecimal representation of user data
7. ASCII representation of user data

Description

file kind

Tester file kind for the RPC request data file.

file name

Specified tester file path name (up to 64 characters).

data number

Sequential data number from the beginning of file (up to 10 digits).

response area size

Response area size (bytes in decimal) specified for the RPC request data file header.

data length

Data length (bytes in decimal) specified for the RPC request data file header.

- Output example with `-k` option = RRQ

```
file kind=RPC request data file (RRQ)
file name=/tmp/rrqfile
-----------------------------------------------------------------
No.1
  response area size=256
  data length=20
  data contents
00000000   52504320 72657175 65737420 64617461 RPC request data
00000010   00000000                            ................
```

(6) Output format (-k option = RRT)

```
file kind=RPC response data file (RRT)                              1.
file name=/tmp/rrtfile
-----------------------------------------------------------------
No.1                                                                2.
  data length=260                                                   3.
  data contents
00000000   52504320 72657370 6f6e7365 20646174 RPC response datu
00000010   61000000 00000000 00000000 00000000 a...............
00000020   00000000 00000000 00000000 00000000 ................     4.
00000020 - 000000ff :     SAME DATA
00000100   00000000
   5.                   6.                             7.
```

Legend:

1. File information
2. Data number
3. Specific information data
4. User data

   The same data is displayed as follows.

   (First matched data location) - (last matched data location) `: SAME DATA`
5. User data location
6. Hexadecimal representation of user data
7. ASCII representation of user data

Description:

file kind

Tester file kind for the RPC response data file.

file name

Specified tester file path name (up to 64 characters).

data number

Sequential data number (up to 10 digits) from the beginning of file.

data length

Data length (bytes in decimal) specified for the RPC response data file header.

- Output example with `-k` option = RRT

```
file kind=RPC response data file (RRT)
file name=/tmp/rrtfile
-----------------------------------------------------------------
No.1
 data length=20
 data contents
00000000   52504320 72657370 6f6e7365 20646174 RPC response data
00000100   61000000                            ...
```

(7) Output format (-k option = XRQ)

```
file kind=XATMI request data file (XRQ)                              ] 1.
file name=/tmp/xrqfile
----------------------------------------------------------------
No.1                                                                 ] 2.
  call kind=call                                                     ] 3.
  flag=0x0000108 (TPNOTRAN)(TPNOCHANGE)
  type=X_OCTET
  subtype=****
  data length=260
  data contents
00000000   74706361 6c6c2058 5f4f4354 45542064 tpcall X_OCTET d  ] 4.
00000010   61746100 00000000 00000000 00000000 ata.............
00000020   00000000 00000000 00000000 00000000 ................
00000030 - 000000ff :     SAME DATA
00000100   00000000
   5.                        6.                        7.
```

Legend:

1. File information
2. Data number
3. Specific information data

4. User data

   The same data is displayed as follows.

   (First matched data location) - (last matched data location) : `SAME DATA`

5. User data location

6. Hexadecimal representation of user data

7. ASCII representation of user data

Description:

file kind

Tester file kind for the XATMI request data file.

file name

Specified tester file path name (up to 64 characters).

data number

Sequential data number (up to 10 digits) from the beginning of file.

call kind

Call kind (up to 7 characters) specified for the XATMI request data file header.

`****` is displayed if no character string is specified.

flag

Flag (8 digits) specified for the XATMI request data file header.

type

Buffer type (up to 8 characters) specified for the XATMI request data file header.

`****` is displayed if no character string is specified.

subtype

Buffer subtype (up to 16 characters) specified for the XATMI request data file header.

`****` is displayed if no character string is specified.

data length

Data length (bytes in decimal) specified for the XATMI request data file header.

- Output example with `-k` option = XRQ

```
file kind=XATMI request data file (XRQ)
file name=/tmp/xrqfile
-----------------------------------------------------------------
No.1
   call kind=call
   flag=0x0000108 (TPNOTRAN)(TPNOCHANGE)
   type=X_OCTET
   subtype=****
   data length=20
   data contents
00000000   74706361 6c6c2058 5f4f4354 45542064 tpcall X_OCTET
00000010   61746100                            data.
```

(8) Output format (-k option = XRT)

```
file kind=XATMI request data file (XRT)                            ] 1.
file name=/tmp/xrtfile
-----------------------------------------------------------------
No.1                                                               ] 2.
   type=X_OCTET                                                    ] 3.
   subtype=****
   rval=0x04000000 (TPFAIL)
   rcode=22
   data length=260
   data contents
00000000   74707265 7475726e 20545046 41494c20 tpreturn TPFAIL     ] 4.
00000010   64617461 00000000 00000000 00000000 data............
00000020   00000000 00000000 00000000 00000000 ................
00000030 - 000000ff :     SAME DATA
00000100   00000000
    5.                    6.                          7.
```

Legend:

1. File information
2. Data number
3. Specific information data
4. User data

   The same data is displayed as follows.

   (First matched data location) - (last matched data location) : `SAME DATA`
5. User data location
6. Hexadecimal representation of user data
7. ASCII representation of user data

Description:

file kind

Tester file kind for the XATMI response data file.

file name

Specified tester file path name (up to 64 characters).

data number

Sequential data number (up to 10 digits) from the beginning of file.

type

Buffer type (up to 8 characters) specified for the XATMI response data file header.

`****` is displayed if no character string is specified.

subtype

Buffer subtype (up to 16 characters) specified for the XATMI response data file header.

`****` is displayed if no character string is specified.

rval

Service termination code (8 digits) specified XATMI response data file header.

rcode

Return code (up to 11 digits in decimal) specified for the XATMI response data file header.

data length

Data length (bytes in decimal) specified for the XATMI response data file header.

- Output example with `-k` option = XRT

```
file kind=XATMI response data file (XRT)
file name=/tmp/xrtfile
-----------------------------------------------------------------
No.1
  type=X_OCTET
  subtype=****
  rval=0x04000000 (TPFAIL)
  rcode=22
  data length=20
  data contents
00000000   74707265 7475726e 20545046 41494c20 tpreturn TPFAIL
00000010   64617461                            data
```

(9) Output format (-k option = XRV)

```
file kind=XATMI receive/send data file (XRV)                            1.
file name=/tmp/xrvfile
----------------------------------------------------------------
No.1                                                                    2.
  type=X_OCTET                                                          3.
  subtype=****
  event flag=0x000000008 (TPEV_SVCSUCC)
  data length=260
  data contents
00000000   74707265 63762072 65637620 64617461 tprecv recv data         4.
00000010   00000000 00000000 00000000 00000000 ................
00000020 - 000000ff :     SAME DATA
00000100   00000000

    5.                    6.                             7.
```

Legend:

1. File information
2. Data number
3. Specific information data
4. User data

   The same data is displayed as follows.

   (First matched data location) - (last matched data location) `: SAME DATA`
5. User data location
6. Hexadecimal representation of user data
7. ASCII representation of user data

Description:

file kind

Tester file kind for the XATMI receive data file and the XATMI send data file.

file name

Specified tester file path name (up to 64 characters).

data number

Sequential data number (up to 10 digits) from the beginning of file.

type

Buffer type (up to 8 characters) specified for the XATMI receive data file header and the XATMI send data file header.

`****` is displayed if no character string is specified.

subtype

Buffer subtype (up to 16 characters) specified for the XATMI receive data file header and the XATMI send data file header.

`****` is displayed if no character string is specified.

event flag

Event flag (8 digits) specified for the XATMI receive data file header and the XATMI send data file header.

data length

Data length (bytes in decimal) specified for the XATMI receive data file header and the XATMI send data file header.

■ Output example with `-k` option = XRV

```
file kind=XATMI receive/send data file (XRV)
file name=/tmp/xrvfile
-----------------------------------------------------------------
No.1
  type=X_OCTET
  subtype=****
  event flag=0x00000008 (TPEV_SVCSUCC)
  data length=20
  data contents
00000000   74707265 63762072 65637620 64617461 tprecv recv data
00000010   00000000                            ....
```

(10) Output format (-k option = NRV)

```
file kind=MCF receive message file (NRV)                          ]1.
file name=/tmp/nrvfile
-----------------------------------------------------------------
No.1                                                              ]2.
  logical terminal name=TERM01                                    ]3.
  map name=MAP01
  segment type=0
  data length=260
  data contents
00000000   4d434620 72656376 206d6573 73616765 MCF recv message   ]4.
00000010   00000000 00000000 00000000 00000000 ................
00000020 - 000000ff :     SAME DATA
00000100   00000000
   5.                  6.                          7.
```

Legend:

1. File information
2. Data number
3. Specific information data
4. User data

   The same data is displayed as follows.

   (First matched data location) - (last matched data location) : `SAME DATA`
5. User data location
6. Hexadecimal representation of user data
7. ASCII representation of user data

Description:

file kind

Tester file kind for the MCF receive message file.

file name

Specified tester file path name (up to 64 characters).

data number

Sequential data number (up to 10 digits) from the beginning of file.

logical terminal name

Logical terminal name (up to 8 characters) specified for the MCF receive message file.

`****` is displayed if no character string is specified.

map name

Map name (up to 8 characters) specified for the MCF receive message file header.

segment type

Segment type (1 character) specified for the MCF receive message file header.

`****` is displayed if no character string is specified.

data length

Data length (bytes in decimal) specified for the MCF receive message file header.

■ Output example with -k option = NRV

```
file kind=MCF receive message file (NRV)
file name=/tmp/nrvfile
------------------------------------------------------------------
No.1
  logical terminal name=TERM01
  map name=MAP01
  segment type=L
  data length=20
  data contents
00000000   4d434620 72656376 206d6573 73616765 MCF recv message
00000010   32000000                            2...
No.2
  logical terminal name=TERM02
  map name=MAP02
  segment type=0
  data length=20
  data contents
00000000   4d434620 72656376 206d6573 73616765 MCF recv message
00000010   33000000                            3...
```

(11) Output format (-k option = COM)

```
file kind=operation command result data file (COM)              ] 1.
file name=/tmp/comfile
------------------------------------------------------------------
No.1                                                             ] 2.
  command result code=1                                          ] 3.
  standard out data length=260
  standard error data length=260
  standard out data content
00000000   7374646f 75742064 61746120 00000000 stdout data ....  ] 4.
00000010   00000000 00000000 00000000 00000000 ................
00000020 - 000000ff :     SAME DATA
00000100   00000000
  standard error data content
00000000   73746465 72722064 61746120 00000000 stderr data ....  ] 5.
00000010   00000000 00000000 00000000 00000000 ................
00000020 - 000000ff :     SAME DATA
00000100   00000000
    6.                     7.                        8.
```

Legend:

1. File information
2. Data number
3. Specific information data
4. Standard output data (user data)

   The same data is displayed as follows.

(First matched data location) - (last matched data location) : `SAME DATA`

5. Standard error output data (user data)

   The same data is displayed as follows.

   (First matched data location) - (last matched data location) : `SAME DATA`

6. User data location
7. Hexadecimal representation of user data
8. ASCII representation of user data

Description:

file kind

Tester file kind for the operating command result data file.

file name

Specified tester file path name (up to 64 characters).

data number

Sequential data number (up to 10 digits) from the beginning of file.

command result code

Command result code (up to 11 digits in decimal) specified for the operating command result data file.

standard out data length

Length (bytes in decimal) of a standard output character string specified for the operating command result data file.

standard error data length

Length (bytes in decimal) of a standard error output character string specified for the operating command result data file.

- Output example with `-k` option = COM

```
file kind=operation command result data file (COM)
file name=/tmp/comfile
-----------------------------------------------------------------
No.1
  command result code=1
  standard out data length=260
  standard error data length=260
  standard out data content
00000000   7374646f 75742064 61746120 00000000 stdout data ....
00000010   00000000                            ....
  standard error data content
00000000   73746465 72722064 61746120 00000000 stderr data ....
00000010   00000000                            ....
```

5.1.5 utols (test status display)

(1) Syntax

utols [*server-name* [*server-name*] ...]

(2) Function

Outputs the status of the test UAP managed by the tester service to standard output. Nothing is output if no test UAP exists.

The `utols` command can only be used when the tester service is active.

(3) Command arguments

- *server-name* ~<identifier of 1-8 characters>

  Specify the name of the user server for which test status is to be displayed.

  When specification is omitted, the status of all the user servers being tested is displayed.

(4) Output format

```
mode    ID      server   service group name     debugger
---------------------------------------------------------
target  usr1    spp01    group1                 dbx
dmyspp  usr1    spp02    group2
simmhp  usr2    spp03    group3                 dbx
usable  ****    spp03    group3
 1.      2.      3.       4.                     5.
```

1. One of the following is displayed as the test mode information for the UAP (value specified in the `test_mode` operand of the user service definition):

`target`

`test_mode=target` specified at startup

`usable`

`test_mode=usable` specified at startup

`dmyspp`

`test_mode=dmyspp` specified at startup

`simmhp`

`test_mode=simmhp` specified at startup

2. Test user ID of the user who started the UAP.

   `****` is displayed when the UAP test mode is `usable`.

3. Server name (up to 8 characters)

4. Service group name (up to 31 characters).

   Nothing is displayed when no service groups are specified.

5. Name (up to 8 characters) of the debugger interlocked to the UAP.

   Nothing is displayed when the UAP is not interlocked to the debugger.

(5) Note

If the OpenTP1 system is immediately shut down or if the test UAP is forcibly terminated while active or inactive, information may be displayed for the inactive or terminated UAP. To display information correctly, restart the UAP for which information was displayed in error.

5.1.6 utomhpsvc (service requests to an MHP)

(1) Syntax

```
utomhpsvc  [-t MCF-receive-message-header-file-name] [-n]
           service-group-name service-name
           MCF-receive-message-file-name
```

(2) Function

Requests the MHP to execute a specified service when using the MCF simulator. The MHP that provides the service must be activated as a simulate MHP linked to the MCF simulation functions library provided by the online tester.

The MHP must be started in test mode; otherwise, a command error occurs. Also, operation is not guaranteed if the service request is made to an SPP running in test mode.

If no reply to the service request is received within the RPC maximum reply-wait time (value specified in the `watch_time` operand in the system common definition), a send/receive timeout condition occurs and the command is not accepted.

(3) Options

- `-t` *MCF-receive-message-header-file-name* ~<1-14 alphanumerics>

  Specify the name of the MCF receive message file containing the header segment to be prefixed to the receive message.

  When specification is omitted, no header segment is prefixed to the receive message.

- `-n`

  Executes the specified service as a non-transaction MHP. When this option is omitted, the service is executed as a transaction MHP.

(4) Command arguments

- *service-group-name* ~<identifier of 1-31 characters>

  Specify the name of the service group to which the service to be executed belongs.

- *service-name* ~<identifier of 1-31 characters>

  Specify the name of the service to be executed.

- *MCF-receive-message-file-name* ~<1-14 alphanumerics>

  Specify the name of the MCF receive message file containing the receive message.

5.1.7 utomsgout (edited output of send messages)

(1) Syntax

```
utomsgout  [{ -i|-r output-file-name }] [-w][{ -o|-l }]
           [-f function-name] [-n number]
           [-t logical-terminal-name]
           [-s service-group-name [,service-name]...]
            MCF-send-message-file-name
```

(2) Function

Edits the send message information output by the online tester and outputs the information to standard output. Or, outputs the information to the specified file when the `-r` option is specified.

A command error occurs if the command is entered while OpenTP1 is writing send messages to the specified MCF send message file.

There are two types of options:

- Options for changing the output format:

  `-i` and `-r`

- Options for selecting output message files:

  `-f`, `-l`, `-n`, `-o`, `-s`, `-t`, and `-w`

When an option with a flag argument is specified more than once, the last specified option is valid.

(3) Options

■ `-i`

Lists send messages in abbreviated form.

This option cannot be specified with the `-r` option.

■ `-r` *output-file-name* ~<pathname>

Specify the name of the file to which the specified messages are to be output. The messages are output in the data format of an MCF receive message file. Therefore, the output file can be used without modification as an MCF receive message file.

This option cannot be specified with the `-i` option. If the `-r` and `-i` options are both omitted, segment information and send message information are output to standard output.

■ `-w`

Edits and outputs only the messages that are not output by the `-r` option.

When this option is omitted, all messages are edited and output.

■ `-o`

Outputs only the oldest message among the editable messages.

This option cannot be specified with the `-l` option. If the `-o` and `-l` options are both omitted, all messages are output.

■ `-l`

Outputs the most recent message among the editable messages.

This option cannot be specified with the `-o` option. If the `-l` and `-o` options are both omitted, all messages are output.

■ `-f` *function-name*

Outputs messages collected for the specified function. The following function names can be specified:

`send`

`dc_mcf_send` function

`reply`

`dc_mcf_reply` function

`execap`

`dc_mcf_execap` function

`sendrecv`

`dc_mcf_sendrecv` function

`sendsync`

`dc_mcf_sendsync` function

The `dc_mcf_resend` function cannot be specified in this option because send messages are not resent (rewritten) by the `dc_mcf_resend` function when the MCF simulator is used.

- `-n` *number*

  Selects output messages by number. To check message numbers, specify the `-i` option to display an abbreviated listing of all send messages.

  This option takes precedence when specified with options other than `-i` or `-r`.

- `-t` *logical-terminal-name* ~<identifier of 1-8 characters>

  Outputs messages sent to the specified logical terminal.

- `-s` *service-group-name* ~<identifier of 1-31 characters>

  service-name ~<identifier of 1-31 characters>

  Outputs messages sent in a specified service. Specify both the service group name and service name, delimiting the two names with a comma (,).

  Two or more services can be specified for a service group. Delimit the service names with commas. Do not insert a space or symbol before or after the comma.

  Both the service group name and service name must be specified. If no service name is specified, the send message information of all the services in the specified service group is edited and output.

  When this option is omitted, send message information is output for all services in all the service groups.

(4) Command argument

- *MCF-send-message-file-name* ~<pathname>

  Specify the name of the MCF send message file containing the send messages.

(5) Output format

(a) -i and -r options omitted

```
time=10:36:12            service group name=group1          1.
message size=20          service name=service1
logical terminal=term01  function=dc_mcf_reply
segment type=L           map name=map01
----------------------------------------------------------------
00000000 5245504c 59313233 34353637 38396162 REPL Y123 4567 89ab
00000010 63646566                             cdef
    :       :        :        :          :     :    :    :    :
   2.                    3.                           4.
```

1. Information on the edited and output send messages:
    - Time at which the messages were collected (hour:minute:second)
    - Message size (up to 10 digits)
    - Logical terminal name (up to 8 characters)
    - Service group name of the sent messages (up to 31 characters).

      `****` is displayed when the service group name is unknown.
    - Service name of the sent messages (up to 31 characters).

      `****` is displayed when the service name is unknown.
    - One of the following function names for which the messages were collected:

      `dc_mcf_send` function

      `dc_mcf_reply` function

      `dc_mcf_execap` function

      `dc_mcf_sendrecv` function

      `dc_mcf_sendsync` function
    - One of the following segment types:

      `M`

      Middle segment

      `L`

      Last segment
    - Map name.

      `****` is displayed when no map name is returned.

2. Relative location
3. Dump display (hexadecimal)
4. ASCII character display.

   A period (.) is displayed when ASCII character display is impossible.

■ Output example

```
time=10:36:12            service group name=group1
message size=20          service name=service1
logical terminal=term01  function=dc_mcf_reply
segment type=L           map name=map01
-----------------------------------------------------------------
00000000  5245504c 59313233 34353637 38396162 REPL Y123 4567 89ab
00000010  63646566                            cdef

time=10:36:13            service group name=group2
message size=10          service name=service2
logical terminal=term01  function=dc_mcf_send
segment type=L           map name=****
-----------------------------------------------------------------
00000000  53454e44 30303030 3030              send 0000 00
                                 :
                                 :
```

(b) -i option specified

```
no    function   service group   service
-----------------------------------------------------------------
 1    reply      group1          service1
 2    send       group2          service2
 :      :          :               :
1.      2.         3.              4.
```

1. Message number in the file
2. Function for which the message was sent:

   `send`

   `dc_mcf_send` function

   `reply`

   `dc_mcf_reply` function

   `execap`

   `dc_mcf_execap` function

   `sendrecv`

`dc_mcf_sendrecv` function

`sendsync`

`dc_mcf_sendsync` function

3. Service group name of the sent messages (up to 31 characters).

   `****` is displayed when the service group name is unknown.

4. Service name of the sent messages (up to 31 characters).

   `****` is displayed when the service name is unknown.

■ Output example

```
no    function   service group    service
-----------------------------------------------------------------
 1     reply    group1          service1
 2     send     group2          service2
 3     execap   group2          ****
 4     send      ****           service3
                           :
                           :
```

(6) Notes

- The send messages collected by the MCF simulation functions are written to the MCF send message file when a function is issued. The messages remain in the file if a rollback occurs.
- When the `-r` option is specified in the `utomsgout` command, the segment type is displayed as `[M...]L`. For example, a logical message consisting of the three segments `F`, `M`, and `L` is actually output as `M`, `M`, `L`. However, this output can be used without modification as input for the online or offline tester.

5.1.8 utosppsvc (service requests to an RPC interface SPP)

(1) Syntax

```
utosppsvc   service-group-name  service-name
            RPC-request-data-file-name
            [RPC-response-data-file-name]
```

(2) Function

Requests an RPC interface SPP to execute a specified service. However, execution of a service cannot be requested for an SPP that expects a transactional RPC (an SPP that requires a transaction to be generated in advance at the UAP making the service request). A command error occurs if the `utosppsvc` command is executed for a UAP other than an RPC interface SPP.

If no reply to the service request is received within the RPC maximum reply-wait time (value specified in the `watch_time` operand in the system common definition), a send/receive timeout condition occurs and the command is not accepted.

This command cannot be used for a simulate MHP.

(3) Command arguments

- *service-group-name* ~<identifier of 1-31 characters>

  Specify the name of the service group to which the service to be executed belongs.

- *service-name* ~<identifier of 1-31 characters>

  Specify the name of the service to be executed.

- *RPC-request-data-file-name* ~<pathname>

  Specify the name of the RPC request data file that contains the input data for the service request.

- *RPC-response-data-file-name* ~<pathname>

  Specify the name of the RPC response data file for storing the response data when the service is executed.

  If this command argument is omitted, the response data is deleted.

  When an existing output file is specified, its contents are overwritten. If the specified file does not exist, the online tester creates the file.

5.1.9 utotrcmrg (merger of UAP trace information)

(1) Syntax

```
utotrcmrg  -o trace-merge-file-name  trace-file-name
           trace-file-name  [trace-file-name  ...]
```

(2) Function

Outputs the trace information in the specified trace files in service execution sequence to a specified file.

Duplicated trace information is output once only.

The merged trace information may not be listed in collection sequence if the merged trace files were collected by different versions of the online tester.

(3) Option

- `-o` *trace-merge-file-name* ~<pathname>

  Specify the name of the trace merge file for output of the merged trace information.

(4) Command argument

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

  Specify the names of the trace files or trace merge files to be merged.

(5) Notes

- A warning message is output if the trace information in a specified trace file is of a version for which nest control is not possible. The trace information is merged by time series.
- A warning message is output if the trace information required for nest control does not exist.

5.1.10 utotrcout (edited output of UAP trace information)

(1) Syntax

```
utotrcout  [-s service-group-name
           [,service-name]...]
           [-v server-name] [-i] [-n]
           [-t [edit-start-date-and-time]
           [,edit-end-date-and-time]] edit-file-name
```

(2) Function

Edits the trace information in the specified trace file or trace merge file and outputs the information to standard output.

(3) Options

- `-s` *service-group-name* ~<identifier of 1-31 characters>

  service-name ~<identifier of 1-31 characters>

  Edits and outputs trace information for a specified service. Specify both the service group name and service name, delimiting the two names with a comma (,).

  Two or more services can be specified for a service group. Delimit the service names with commas. Do not insert a space or symbol before or after the comma.

  Both the service group name and service name must be specified. If no service name is specified, trace information is edited and output to standard output for all the services in the specified service group.

  When this option is omitted, the trace information of all the service groups in the specified file is edited and output.

  If this option is specified with the `-v` option, both specifications apply to the output trace information.

  If this option is specified with the `-n` option, trace information for the service request destination is also output.

- `-v` *server-name* ~<identifier of 1-8 characters>

  Edits and outputs trace information for the specified server.

  When this option is omitted, the trace information of all the servers in the specified file is edited and output.

  If this option is specified with the `-s` option, both specifications apply to the output trace information.

  If this option is specified with the `-n` option, trace information on service requests to the specified server is also output.

- `-i`

  Outputs selected information, such as function names, from the trace information collected in the specified file to standard output.

  When this option is omitted, all trace information is output to standard output.

- `-n`

  Outputs the trace information collected in the specified file to standard output in the sequence in which the information was collected.

- `-t` *edit-start-date-and-time*`,` *edit-end-date-and-time*

  Sets the time range for output of trace information. The specified start time is corrected to the log time for the process that made the first service request.

  Specify the start and end times within the range from 0:0:0 on January 1, 1970 to the current time.

  If the edit start time is omitted, trace information is output from the start of the specified file up to the specified edit end time.

  If the edit end time is omitted, trace information is output from the specified edit start time up to the end of the specified file.

  Specify the start and end times in the following format:

  hhmmss`[`*MMDD*`[`*YYYY*`]]`

  where

  hh

  hour ($00 \leq hh \leq 23$)

  mm

  minute ($00 \leq mm \leq 59$)

  ss

second (00 ≤ *ss* ≤ 59)

MM

month (01 ≤ *MM* ≤ 12)

DD

day (01 ≤ *DD* ≤ 31)

YYYY

year (1970 ≤ *YYYY* ≤ 9999)

If *YYYY* is omitted in the start or end time, the current year is assumed. If *MM*, *DD*, and *YYYY* are all omitted, the current month, day, and year are assumed.

Either the edit start time or the edit end time must be specified.

(4) Command argument

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

  Specify the name of the trace file or trace merge file to be edited.

(5) Output format

(a) -i option omitted

```
SERVER NAME          = UTOSPP01
                           EDITION OBJECT DATE AND TIME = 98/01/27 11:17:40
SERVICE GROUP NAME = UTOSPP01
    COLLECTION DATE AND TIME = 98/01/26 14:50:28                              1.
    COLLECTION NO. = 1
      PROCESS ID   = 1925
      TEST USER ID = usr1

FUNCTION = dc_rpc_open (ENTRANCE)
    COLLECTION DATE AND TIME = 98/01/26 14:50:29
    COLLECTION NO. = 2      SERVICE NAME = ****
                                          :                                   2.
                                                                        3.
                                          :

                                          :
```

1. Tester information:
   - Name of the server at which the UAP was started (up to 8 characters)
   - Date and time, corrected to the log time for the process that made the first service request

(last two digits of year/month/day hour:minute:second)

- Service group name of the activated service (up to 31 characters).

  `****` is displayed for an SUP.

- Time at which the UAP trace information was collected

  (last two digits of year/month/day hour:minute:second)

- Sequence number of the entry for which trace information was collected (6 digits)

- ID of the process for which trace information was collected

- Test user ID of the user who started the UAP (up to 4 characters)

2. UAP trace information (same output format as for `uatdump -e` command):

   - Type of trace information collected

   - Date and time when the trace information was collected.

     Not displayed for functions that activate or terminate service requests.

   - Date and time when the tester information or the UAP trace information was collected in the format of year (last two digits)/month/day hour:minutes:seconds.

   - Sequential number (six digits) of the entry that collected trace information

   - Name of the service that activated the UAP (up to 31 characters).

     `****` is displayed for an SUP or when the service is unknown.

3. Output area for call information on OpenTP1 functions

■ Output example (-i option omitted)

```
SERVER NAME        = sppni01
                           EDITION OBJECT DATE AND TIME = 98/03/08 16:22:42
SERVICE GROUP NAME = sppni01
    COLLECTION DATE AND TIME = 98/03/08 16:22:42
    COLLECTION NO. = 1
      PROCESS ID   = 3895
      TEST USER ID = dam

FUNCTION = dc_rpc_open (ENTRANCE)
    COLLECTION DATE AND TIME = 98/03/08 16:22:42
    COLLECTION NO. = 2      SERVICE NAME = ****
      SERVER NAME = sppni01
      OPTION FLAG = 0x00000000 (DCNOFLAGS)                              1.

FUNCTION = dc_rpc_open (EXIT)
    COLLECTION DATE AND TIME = 98/03/08 16:22:43
    COLLECTION NO. = 3      SERVICE NAME = ****
      SERVER NAME = sppni01
      OPTION FLAG = 0x00000000 (DCNOFLAGS)
    RETURN CODE = 0  (NORMAL TERMINATION)

FUNCTION = dc_rpc_mainloop (ENTRANCE)
    COLLECTION DATE AND TIME = 98/03/08 16:22:43
    COLLECTION NO. = 4      SERVICE NAME = ****
      SERVER NAME = sppni01
      OPTION FLAG = 0x00000000 (DCNOFLAGS)
```

```
SERVER NAME         = supni01
                            EDITION OBJECT DATE AND TIME = 98/03/08 16:22:45
SERVICE GROUP NAME = ****
    COLLECTION DATE AND TIME = 98/03/08 16:22:45
    COLLECTION NO. = 1
      PROCESS ID    = 3898
      TEST USER ID = dam

FUNCTION = dc_rpc_open (ENTRANCE)
    COLLECTION DATE AND TIME = 98/03/08 16:22:45
    COLLECTION NO. = 2       SERVICE NAME = ****
      SERVER NAME = supni01
      OPTION FLAG = 0x00000000 (DCNOFLAGS)

FUNCTION = dc_rpc_open (EXIT)
    COLLECTION DATE AND TIME = 98/03/08 16:22:45
    COLLECTION NO. = 3       SERVICE NAME = ****
      SERVER NAME = supni01
      OPTION FLAG = 0x00000000 (DCNOFLAGS)
    RETURN CODE = 0  (NORMAL TERMINATION)

FUNCTION = dc_adm_complete (ENTRANCE)
    COLLECTION DATE AND TIME = 98/03/08 16:22:45
    COLLECTION NO. = 4       SERVICE NAME = ****
      OPTION FLAG = 0x00000000 (DCNOFLAGS)

FUNCTION = dc_adm_complete (EXIT)
    COLLECTION DATE AND TIME = 98/03/08 16:22:45
    COLLECTION NO. = 5       SERVICE NAME = ****
      OPTION FLAG = 0x00000000 (DCNOFLAGS)
    RETURN CODE = 0 (DC_OK)
```

2.

```
FUNCTION = dc_rpc_call (ENTRANCE)
    COLLECTION DATE AND TIME = 98/03/08 16:22:45
    COLLECTION NO. = 6       SERVICE NAME = ****
  SERVICE GROUP NAME OF CALLED SERVICE = sppni01
      NAME OF CALLED SERVICE = svccal
      SEND DATA LENGTH(1024)
      ------ SEND DATA ------
      000078    00000000 00000001 00000001 00000000   .... .... .... ....
      000088    00000000 00000073 70706e69 30320000   .... ...s ppni 02..
      000098    00000000 00000000 00000000 00000000   .... .... .... ....
      0000a8    00000000 00000073 76636461            .... ...s vcda
      RECEIVE DATA LENGTH(1024)
      OPTION FLAG = 0x00000000 (DCNOFLAGS)
```

2.

```
SERVER NAME        = sppni01
                           EDITION OBJECT DATE AND TIME = 98/03/08 16:22:45
SERVICE GROUP NAME = sppni01
    COLLECTION DATE AND TIME = 98/03/08 16:22:45
    COLLECTION NO. = 5
      PROCESS ID   = 3895
      TEST USER ID = dam

FUNCTION = STARTING SERVICE FUNCTION
    COLLECTION DATE AND TIME = 98/03/08 16:22:45
    COLLECTION NO. = 6      SERVICE NAME = svccal
      CALLING NODE NAME = 2C3G009
      CALLING SERVICE GROUP NAME = ****
      CALLING SERVICE NAME = ****
      INPUT MESSAGE LENGTH(1024)
      ------ INPUT DATA ------
      000098    00000000 00000001 00000001 00000000   .... .... .... ....
      0000a8    00000000 00000073 70706e69 30320000   .... ...s ppni 02..
      0000b8    00000000 00000000 00000000 00000000   .... .... .... ....
      0000c8    00000000 00000073 76636461            .... ...s vcda

FUNCTION = dc_trn_begin (ENTRANCE)
    COLLECTION DATE AND TIME = 98/03/08 16:22:45
    COLLECTION NO. = 7      SERVICE NAME = svccal

FUNCTION = dc_trn_begin (EXIT)
    COLLECTION DATE AND TIME = 98/03/08 16:22:45
    COLLECTION NO. = 8      SERVICE NAME = svccal
      ------ XID ------
      000034    01030000 00000018 00000024 00000013   .... .... ...$ ....
      000044    00000f37 40404040 5075746f 00000000   ...7 @@@@ Puto ....
      000054    00000000 00000001 00000013 00000001   .... .... .... ....
      000064    ffffffff 40404040 5075746f 00000f37   .... @@@@ Puto ...7
      000074    00000000 00000000 00000000 00000000   .... .... .... ....
      000084 - 0000a4 :      SAME DATA
      0000b4    00000000 00000000 00000000            .... .... ....
    RETURN CODE = 0 (DC_OK)

FUNCTION = dc_dam_open (ENTRANCE)
    COLLECTION DATE AND TIME = 98/03/08 16:22:45
    COLLECTION NO. = 9      SERVICE NAME = svccal
      REQUEST CODE = OPEN
      LOGICAL FILE NAME = TAMTABLE
      OPTION FLAG = 0x00000002 (DCDAM_BLOCK_EXCLUSIVE)

FUNCTION = dc_dam_open (EXIT)
    COLLECTION DATE AND TIME = 98/03/08 16:22:45
    COLLECTION NO. = 10      SERVICE NAME = svccal
      REQUEST CODE = OPEN
      LOGICAL FILE NAME = TAMTABLE
      OPTION FLAG = 0x00000002 (DCDAM_BLOCK_EXCLUSIVE)
    RETURN CODE = 16842753  (FILE DESCRIPTOR)
```

3.

```
FUNCTION = dc_dam_write (ENTRANCE)
    COLLECTION DATE AND TIME = 98/03/08 16:22:46
    COLLECTION NO. = 11      SERVICE NAME = svccal
      REQUEST CODE = WRIT
      FILE IDENTIFIER = 16842753
      OPTION FLAG = 0x00000000 (DCNOFLAGS)
      KEY COUNT = 1                      BUFFER LENGTH = 512
      RELATIVE BLOCK NUMBER = 3
      000054    00000000 00000000 00000000 00000000   .... .... .... ....
      000064 - 0000e4 :       SAME DATA
      0000f4    00000000 00000000

FUNCTION = dc_dam_write (EXIT)
    COLLECTION DATE AND TIME = 98/03/08 16:22:46
    COLLECTION NO. = 12      SERVICE NAME = svccal
      REQUEST CODE = WRIT
      LOGICAL FILE NAME = TAMTABLE
      FILE IDENTIFIER = 16842753
      OPTION FLAG = 0x00000000 (DCNOFLAGS)
      KEY COUNT = 1                      BUFFER LENGTH = 512
      RELATIVE BLOCK NUMBER = 3
      000054    00000003 7069643D 33383936 20757064   .... pid= 3896  upd
      000064    6174655f 636f756e 743d3100 00000000   ate_ coun t=1. ....
      000074    00000000 00000000 00000000 00000000   .... .... .... ....
      000084 - 0000e4 :       SAME DATA
      0000f4    00000000 00000000                     .... ....
    RETURN CODE = 0 (DC_OK)

FUNCTION = dc_dam_close (ENTRANCE)
    COLLECTION DATE AND TIME = 98/03/08 16:22:46
    COLLECTION NO. = 13      SERVICE NAME = svccal
      REQUEST CODE = CLOS
      FILE IDENTIFIER = 16842753
      OPTION FLAG = 0x00000000 (DCNOFLAGS)

FUNCTION = dc_dam_close (EXIT)
    COLLECTION DATE AND TIME = 98/03/08 16:22:46
    COLLECTION NO. = 14      SERVICE NAME = svccal
      REQUEST CODE = CLOS
      LOGICAL FILE NAME = TAMTABLE
      FILE IDENTIFIER = 16842753
      OPTION FLAG = 0x00000000 (DCNOFLAGS)
    RETURN CODE = 0 (DC_OK)

FUNCTION = dc_trn_unchained_commit (ENTRANCE)
    COLLECTION DATE AND TIME = 98/03/08 16:22:46
    COLLECTION NO. = 15      SERVICE NAME = svccal

FUNCTION = dc_trn_unchained_commit (EXIT)
    COLLECTION DATE AND TIME = 98/03/08 16:22:46
    COLLECTION NO. = 16      SERVICE NAME = svccal
    RETURN CODE = 0 (DC_OK)

FUNCTION = ENDING SERVICE FUNCTION
    COLLECTION DATE AND TIME = 98/03/08 16:22:46
    COLLECTION NO. = 17      SERVICE NAME = svccal
      CALLING NODE NAME = 2C3G009
      CALLING SERVICE GROUP NAME = ****
      CALLING SERVICE NAME = ****
      OUTPUT MESSAGE LENGTH(13)
      ------ OUTPUT MESSAGE ------
      000098    7376636e 6964616d 20656e64 00          svcn idam  end .
```

3.

```
SERVER NAME         = supni01
                           EDITION OBJECT DATE AND TIME = 98/03/08 16:22:46
SERVICE GROUP NAME = ****
    COLLECTION DATE AND TIME = 98/03/08 16:22:46
    COLLECTION NO. = 7
      PROCESS ID    = 3898
      TEST USER ID = dam

FUNCTION = dc_rpc_call (EXIT)
    COLLECTION DATE AND TIME = 98/03/08 16:22:46
    COLLECTION NO. = 8      SERVICE NAME = ****
  SERVICE GROUP NAME OF CALLED SERVICE = sppni01
      NAME OF CALLED SERVICE = svccal**
      SEND MESSAGE LENGTH(1024)
      ------ SEND DATA ------
      000078    00000000 00000001 00000001 00000000   .... .... .... ....
      000088    00000000 00000073 70706e69 30320000   .... ...s ppni 02..
      000098    00000000 00000000 00000000 00000000   .... .... .... ....
      0000a8    00000000 00000073 76636461            .... ...s vcda
      RECEIVE DATA LENGTH(13)
      ------ RECEIVE DATA ------
      0000b8    7376636e 6964616d 20656e64 00          svcn idam  end .
      OPTION FLAG = 0x00000000 (DCNOFLAGS)
    RETURN CODE = 0  (NORMAL TERMINATION)

FUNCTION = dc_rpc_close (ENTRANCE)
    COLLECTION DATE AND TIME = 98/03/08 16:22:46
    COLLECTION NO. = 9      SERVICE NAME = ****
      OPTION FLAG = 0x00000000 (DCNOFLAGS)

FUNCTION = dc_rpc_close (EXIT)
    COLLECTION DATE AND TIME = 98/03/08 16:22:46
    COLLECTION NO. = 10      SERVICE NAME = ****
      OPTION FLAG = 0x00000000 (DCNOFLAGS)
```

4.

```
SERVER NAME         = sppni01
                           EDITION OBJECT DATE AND TIME = 98/03/08 16:27:37
SERVICE GROUP NAME = sppni01
    COLLECTION DATE AND TIME = 98/03/08 16:27:37
    COLLECTION NO. = 18
      PROCESS ID   = 3895
      TEST USER ID = dam

FUNCTION = dc_rpc_mainloop (EXIT)
    COLLECTION DATE AND TIME = 98/03/08 16:27:37
    COLLECTION NO. = 19      SERVICE NAME = ****
      OPTION FLAG = 0x00000000 (DCNOFLAGS)
    RETURN CODE = 0  (NORMAL TERMINATION)

FUNCTION = dc_rpc_close (ENTRANCE)
    COLLECTION DATE AND TIME = 98/03/08 16:27:37
    COLLECTION NO. = 20      SERVICE NAME = ****
      OPTION FLAG = 0x00000000 (DCNOFLAGS)

FUNCTION = dc_rpc_close (EXIT)
    COLLECTION DATE AND TIME = 98/03/08 16:27:37
    COLLECTION NO. = 21      SERVICE NAME = ****
      OPTION FLAG = 0x00000000 (DCNOFLAGS)
```

5.

1. Trace information collected at SPP startup
2. Trace information collected at SUP startup
3. Trace information collected at service execution
4. Trace information collected at SUP completion
5. Trace information collected at SPP completion

(b) -i option specified

```
98/01/26 14:48:37  <98/01/26 15:50:28>                              <1925>
UTOSPP01 (UTOSPP01) <0>
dc_rpc_open (ENTRANCE)
        :
        :
```

1.

2.

1. Tester information:
    - Date and time, corrected to the log time for the process that made the first service request

      (last two digits of year/month/day hour:minute:second)

- Date and time when the tester information was collected

  (last two digits of year/month/day hour:minute:second)
- ID of the process for which the UAP trace was collected
- Name of the server at which the UAP was started (up to 8 characters)
- Service group name of the activated service (up to 31 characters).

  **** is displayed for an SUP.
- Nest number of the UAP for which trace information was collected.

  0 is displayed for trace information for which the online tester version cannot perform nest control.

  When a simulated client UAP or TP1/Client UAP makes the service request, the nest numbers of the service request destinations are displayed from 1.

2. UAP trace information
   - Type of trace information collected
   - Time at which the trace information was collected.

     For functions that activate or terminate service requests, the name of the service that activated the UAP is displayed (up to 31 characters).

■ Output example (-i option specified)

```
98/03/08 16:22:42  <98/03/08 16:22:42>                                      <3895>
sppni01 (sppni01) <0>                                                              1.
dc_rpc_open (ENTRANCE)
dc_rpc_open (EXIT)
dc_rpc_mainloop (ENTRANCE)

98/03/08 16:22:45  <98/03/08 16:22:45>                                      <3898>
supni01 (****) <0>
dc_rpc_open (ENTRANCE)
dc_rpc_open (EXIT)                                                                 2.
dc_adm_complete (ENTRANCE)
dc_adm_complete (EXIT)
dc_rpc_call (ENTRANCE)

98/03/08 16:22:42  <98/03/08 16:22:42>                                      <3895>
sppni01 (sppni01) <1>
STARTING SERVICE FUNCTION (svccal)
dc_trn_begin (ENTRANCE)
dc_trn_begin (EXIT)
dc_dam_open (ENTRANCE)
dc_dam_open (EXIT)
dc_dam_write (ENTRANCE)                                                            3.
dc_dam_write (EXIT)
dc_dam_close (ENTRANCE)
dc_dam_close (EXIT)
dc_trn_unchained_commit (ENTRANCE)
dc_trn_unchained_commit (EXIT)
ENDING SERVICE FUNCTION

98/03/08 16:22:46  <98/03/08 16:22:46>                                      <3898>
supni01 (****) <0>
dc_rpc_call (EXIT)                                                                 4.
dc_rpc_close (ENTRANCE)
dc_rpc_close (EXIT)

98/03/08 16:22:46  <98/03/08 16:22:46>                                      <3895>
sppni01 (sppni01) <0>
dc_rpc_mainloop (EXIT)                                                             5.
dc_rpc_close (ENTRANCE)
dc_rpc_close (EXIT)
```

1. Trace information collected at SPP startup
2. Trace information collected at SUP startup
3. Trace information collected at service execution
4. Trace information collected at SUP completion
5. Trace information collected at SPP completion

(6) Notes

- When the specified edit file contains trace information of an older version than this command, a warning message is output and the information is output in the order in which it was stored in the file.
- When the `-n` option is specified, a warning message is output if the required trace information does not exist (part of the information is missing).
- When the `-n` option is specified and the edition start date and time specified in the `-t` option is a time between the edition object date and times for two consecutive groups, the trace information for the latter and subsequent groups is output.
- When the `-n` option is specified, trace information is output up to the nesting level of the client process, even if the edition object date and time of the trace information for the service request destination exceeds the edition end date and time specified in the `-t` option.
- When trace information is collected with the complete I/O data specified, tester information may be output in the middle of I/O data.
- Valid option combinations are shown below.

Specifiable option combinations	-s	-v	-n	-t	-i
`-s`	`-s`	`-s`, `-v`	`-s`, `-n`	`-s`, `-t`	`-s`, `-i`
`-v`	--	`-v`	`-v`, `-n`	`-v`, `-t`	`-v`, `-i`
`-n`	--	--	`-n`	`-n`, `-t`	`-n`, `-i`
`-t`	--	--	--	`-t`	`-t`, `-i`
`-i`	--	--	--	--	`-i`

Legend:

`-`*x*: Only the `-`*x* option is valid.

`-`*x*, `-`*y*: Both the `-`*x* and `-`*y* options are valid.

--: Not applicable

5.1.11 utoxsppsvc (service requests to an XATMI interface SPP)

(1) Syntax

utoxsppsvc [-f *send/receive-control-file-name*]
service-name *typed-buffer-definition-file-name*
XATMI-request-data-file-name
[*XATMI-response-data-file-name*]

(2) Function

Requests an XATMI interface SPP to execute a specified service. A command error occurs if the `utoxsppsvc` command is executed for a UAP other than an SPP that uses XATMI.

This command cannot be used for a simulate MHP.

(3) Options

■ `-f` *send/receive-control-file-name* ~<pathname>

For an interactive service request, specify the name of the send/receive control file that defines the send and receive procedures.

(4) Command arguments

■ *service-name* ~<identifier of 1-31 characters>

Specify the name of the service to be executed.

■ *typed-buffer-definition-file-name* ~<pathname>

Specify the name of the typed buffer definition file that defines typed buffer information.

■ *XATMI-request-data-file-name* ~<pathname>

Specify the name of the XATMI request data file that contains the input data passed when a service is requested (when connection is established).

■ *XATMI-response-data-file-name* ~<pathname>

Specify the name of the XATMI response data file for storing the receive data during service execution and the response data after service execution.

If this command argument is omitted, the response data is deleted.

When an existing output file is specified, its contents are overwritten. If the specified file does not exist, the online tester creates the file.

(5) Notes

- Only one service request can be made interactively.
- Set the type of service request function (request/response or conversational service paradigm) in `call_kind` in the XATMI request data file.
- The `-f` option is ignored if specified for the request/response service paradigm. If the `-f` option is omitted for the conversational service paradigm, a command error occurs.
- Service requests from within a transaction cannot be simulated.
- No error occurs if the send/receive control file contains no valid lines, but the

command terminates immediately after the request for establishing connection.

- If no receive data or response data is received, no XATMI response data file is created. Provided at least one item of data is received, the file is created even if an error subsequently occurs. The data up to the error remains in the file.
- The data in the XATMI receive or XATMI response data file is invalidated if the data length specified for the file differs from the typed buffer length specified in the typed buffer definition file.
- The following conditions occur if the buffer length managed by the SPP differs from the typed buffer length specified in the typed buffer definition file:

  1. Service request error at `utoxsppsvc` command execution
  2. Data receive error at `utoxsppsvc` command execution
  3. Data receive error in the SPP

- If an XATMI response data file already exists, its contents are deleted when the `utoxsppsvc` command starts. No data remains in the file, even if no data is output.

Chapter

6. Error Recovery

This chapter explains the errors related to online tester operation and how to handle them.

This chapter contains the following section:

6.1 Handling online tester errors

6.1 Handling online tester errors

This chapter describes how to handle online tester errors. See the manual *OpenTP1 Operation* for details on errors not related to the online tester.

6.1.1 Error conditions and causes

Table 6-1 lists the types of errors that may occur with the online tester and their probable causes.

Table 6-1: Online tester errors and causes

Error	Cause	Manual reference
Online tester command does not terminate normally.	No online tester definition in system definition.	6.1.2 (1)
	Incorrect option or command argument.	
	File for command execution cannot be accessed.	
UAP trace information not collected.	No test target specified in system definition.	6.1.2 (2)
	Zero specified in `max_trace_file_size` operand in the system definition.	
	Zero specified in `UAP_trace_max` operand in the system definition.	
Send messages not collected.	No test target specified in system definition.	6.1.2 (3)
	Zero specified in `max_message_file_size` operand in the system definition.	
	Number of send messages exceeds the upper limit.	
No send data collected for interactive service requests.	Output of send data not defined in the system definition.	6.1.2 (4)
Test UAP does not start.	Test user ID not set.	6.1.4 (1)
	No test target specified in system definition.	
UAP does not start in non-test mode.	Specified as a test UAP in the system definition.	6.1.4 (2)
`dc_rpc_open` function in a test UAP returns an error.	No online tester definition in the system definition.	6.1.4 (3)
Test UAP restarted with `_uto` as the test user ID.	Conflict in UAP status control between the OpenTP1 system and the online tester.	6.1.4 (4)

Error	Cause	Manual reference
Test UAP does not recover after abnormal termination.	Test UAP interlocked to the debugger.	6.1.4 (5)
Debugger-interlocked UAP frequently causes a time-out error.	Incorrect value specified for the monitoring time in the user service definition.	6.1.4 (6)

6.1.2 Online tester errors

The following explains how to handle online tester errors.

(1) Online tester command does not terminate normally

Take one of the following actions, then re-enter the command:

- If usage of the online tester is not specified in the system service configuration definition, terminate OpenTP1, configure the online tester in the definition (specify `Y` in the `uto_conf` operand), then restart OpenTP1.
- If an option or command argument is incorrectly specified, correct the option or command argument.
- If a file required for command execution does not exist, create the file. Or, if the existing file cannot be used because access is prohibited, change the access authority.

(2) UAP trace information not collected

(a) No UAP trace information collected at all

If zero is specified as the maximum size of the trace file (`max_trace_file_size` operand) in the tester service definition, terminate OpenTP1, specify 1 or a higher value, then restart OpenTP1.

(b) Trace information not collected for a specific UAP

Take one of the following actions:

- If the UAP is not specified as a test UAP in the user service definition, terminate the UAP, correct the definition (specify `target` in the `test_mode` operand), then restart the UAP.
- If zero is specified as the maximum number of UAP traces (`uap_trace_max` operand) in the user service definition, terminate the UAP, specify 1 or a higher value, then restart the UAP.

(c) Some trace information missing

Take one of the following actions:

- If the trace file is full, back up to another file, then delete the full file.
- If the online tester shut down during UAP execution, restart the online tester, then re-execute the UAP.
- If the UAP detected an abnormality and immediately shut down without collecting the core file, modify the program so that the core file can be collected, then re-execute the UAP.

(3) Send messages not collected

(a) No send messages collected at all

If zero is specified as the maximum size of the MCF send message file (`max_message_file_size` operand) in the tester service definition, terminate OpenTP1, specify 1 or a higher value, then restart OpenTP1.

(b) Send messages not collected for a specific UAP

If the UAP is not specified as a test UAP in the user service definition, terminate the UAP, correct the definition (specify `target` in the `test_mode` operand), then restart the UAP.

(c) Some send messages missing

If the MCF send message file is full, back up to another file, then delete the full file.

(4) No send data collected for interactive service requests.

(a) No send data collected at all

If `N` is specified for send data output (`test_xatmi_send_file` operand) in the user service definition, terminate the UAP, specify `Y`, then restart the UAP.

6.1.3 File errors

If an error occurs in a file created by the online tester, check the cause of the error from the file name and error code displayed in the error message, and take appropriate action.

6.1.4 UAP errors

The following explains how to handle UAP errors.

(1) Test UAP does not start

Take one of the following actions, then re-start the UAP:

- Set the test user ID if omitted.
- Reset the test user ID if incorrect.
- If the UAP is not specified as a test UAP in the user service definition, terminate the UAP, correct the definition (specify `target` in the `test_mode` operand), then

restart the UAP.

(2) UAP does not start in non-test mode

If the UAP is specified as a test target in the user service definition, specify the UAP as a non-test UAP (specify `no` in the `test_mode` operand) or delete the definition statement. Then restart the UAP.

(3) dc_rpc_open function in a test UAP returns an error

If usage of the online tester is not specified in the system service configuration definition, terminate OpenTP1, configure the online tester in the definition (specify `Y` in the `uto_conf` operand), then restart OpenTP1.

(4) Test UAP restarted with _uto as the test user ID

`_uto` may be set as the test user ID when a test UAP is restarted after:

- Forced termination of the OpenTP1 system or UAP during normal termination processing of the UAP
- Abnormal termination of the OpenTP1 system

In these cases, a message reports that `_uto` was set as the test user ID at system restart. To execute the UAP with a different test user ID, terminate and then restart the UAP.

(5) Test UAP does not recover after abnormal termination

When the test UAP is interlocked to the debugger, the UAP is not recovered if it terminates abnormally. A message appears, notifying the UAP recovery is disabled. To restart this UAP, stop the debugger if the debugger process remains.

(6) Debugger-interlocked UAP causes a time-out error frequently

A debugger-interlocked UAP may frequently cause a time-out error depending on a value specified for the monitoring time in the user service definition. Table 6-2 shows time-out error events and related definitions.

Table 6-2: Time-out error events caused by a debugger-interlocked UAP and related definitions

Time-out error event	Set format of the related user service definition
`dc_rpc_call` function times out and returns an error.	`set watch_time`
A time-out error abnormally terminates the corresponding transaction branch process, activating the recovery process. The UAP is then terminated forcibly.	`set trn_expiration_time` `set trn_cpu_time`
A time-out error abnormally terminates the corresponding UAP. The UAP is terminated forcibly.	`set watch_next_chain_time`

Time-out error event	Set format of the related user service definition
A time-out error abnormally terminates the corresponding UAP without shutdown by the service group. The UAP is terminated forcibly.	`set term_watch_time`

Chapter
7. Facilities

This chapter describes the following facilities provided by the MCF online tester:

Disabling updating of non-MCF resources

Invalidating send messages

Invalidating application startup messages

Suppressing error events

Suppressing MHP automatic shutdown

Collecting UAP trace information

This chapter contains the following sections:

7.1 MHP testing
7.2 Collecting test information

7.1 MHP testing

The MCF online tester allows the user to test and check the operation of a newly created MHP or modified MHP. Both of the following conditions must be satisfied for a test to be performed:

- The MHP must be within a transaction.
- The MHP must be activated directly from TP1/Message Control.

To test an MHP, enter the `mcfutfst` command to declare use of the MCF online tester and to access its facilities. Then, enter the `mcftules`, `mcfauaps` or `mcftusgs` command to start testing. Or, to check whether the facilities of the MCF online tester are available, enter the `mcflsutf` command.

Tests can be performed on:

- A logical terminal
- An application
- A service group

The test start command differs in each case. Use the `mcftules` command to test a logical terminal. The test runs from the time a message from the specified logical terminal is received by the application until no further test messages remain.

Use the `mcfauaps` command to test an application. The test runs from the time the specified application receives a message until no further test messages remain. The type of application (user application or MCF event) can be selected by specifying the `-k` operand in the `mcfauaps` command.

Use the `mcftusgs` command to test a service group. The test runs while the service group specified by the `mcftusgs` command is active.

The MCF online tester facilities to be used in a test can also be specified as options in the test start commands. The specifiable facilities are described below.

7.1.1 Disabling updating of non-MCF resources

When a test MHP updates resources managed by another resource manager during message processing, the updated resources can be restored to their previous status at completion of the transaction. This facility means that the user does not need to restore the resources after testing.

To use this facility, specify `backout` in the `-e` option of the test start command.

7.1.2 Invalidating send messages

Messages sent by a test MHP can be invalidated, allowing the MHP to be tested without affecting online jobs.

This facility invalidates messages sent by the following functions issued by the test MHP:

- `dc_mcf_send` function (message send)
- `dc_mcf_sendsync` function (synchronous message send)
- `dc_mcf_resend` function (message resend)

To use this facility, specify `swmsg` in the `-e` option of the test start command.

Messages sent by the following functions cannot be invalidated:

- `dc_mcf_reply` function (response message send)
- `dc_mcf_sendrecv` function (synchronous message send and receive)

7.1.3 Invalidating application startup messages

Startup messages for branch applications can be invalidated, allowing the MHP to be tested without affecting online jobs.

To use this facility, specify `execap` in the `-e` option of the test start command.

Application startup messages for response messages cannot be invalidated.

7.1.4 Suppressing error events

Error events generated in a test MHP can be suppressed, allowing the MHP to be tested without affecting online jobs.

The following error events can be suppressed:

- `ERREVT1` (MCF event that reports detection of an invalid application name, suppressed for logical terminal tests only)
- `ERREVT2` (MCF event that reports discarding of a message at abnormal termination before issue of the `dc_mcf_receive` function)
- `ERREVT2` (MCF event that reports discarding of a message generated at automatic shutdown)
- `ERREVT3` (MCF event that reports UAP abnormal termination at abnormal termination during MHP execution)

To use this facility, specify `errevt` in the `-e` option of the test start command.

The following error events cannot be suppressed:

- `ERREVTA` (MCF event that reports discarding of an unprocessed message)
- `ERREVT4` (MCF event that reports discarding of a timer-start message)

7.1.5 Suppressing MHP automatic shutdown

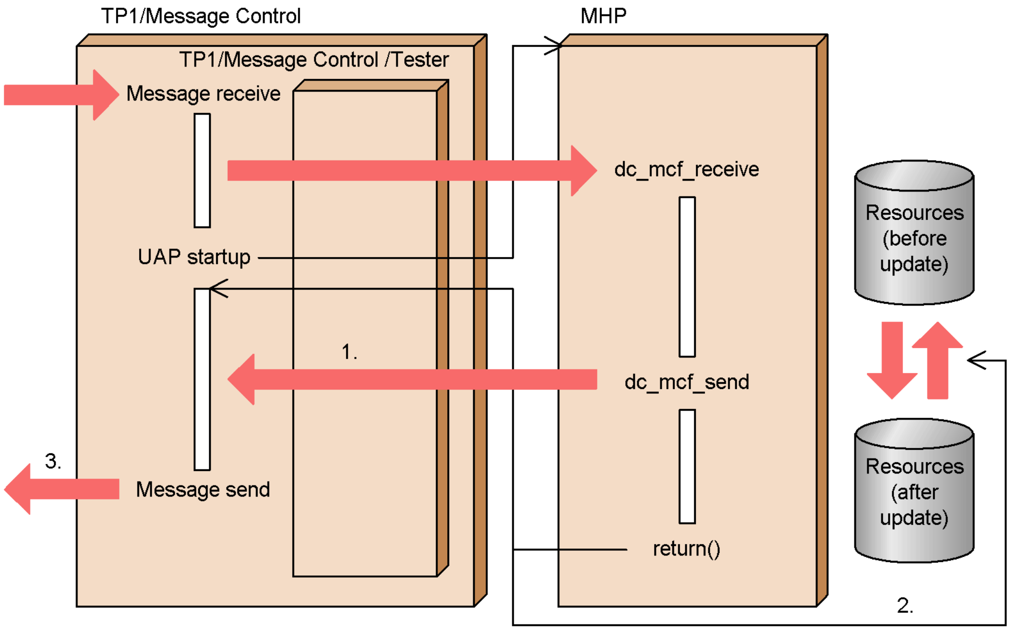
The MHP normally shuts down automatically at abnormal termination. Automatic shutdown can be suppressed so that the user does not need to enter an operating command to release shutdown status.

To use this facility, specify `holdlimit` in the `-e` option of the test start command.

After the test start command has been executed, applications can be started in the same way as when not using the MCF online tester.

Figure 7-1 shows an example of transaction processing from receiving to sending a message. If the MHP terminates abnormally, MCF resources are restored to their status before the transaction began.

Figure 7-1: Example of transaction processing from message receive to message send



Explanation:

1. Messages sent during a test are handled as follows:

Message type	-e option of test start command	
	swmsg specified	swmsg omitted
Inquiry response messages	Sent	Sent

Message type	-e option of test start command	
	swmsg specified	swmsg omitted
Branch messages	Not sent#	Sent

#: The interface is checked at message send and an error status code is returned if an error occurs.

2. Resources of resource managers other than the MCF are restored to their previous status when `backout` is specified in the `-e` option of the test start command. Updated resources are not restored when `backout` is omitted.
3. The send message is output if one exists.

7.2 Collecting test information

7.2.1 Collecting UAP trace information

Trace information can be collect for a test MHP so that MHP operation can be checked. However, the TP1/Server Base online tester must also be used.

To use this facility, specify the test user ID in the `-u` option of the command for the MCF online tester use declaration (`mcfutfst`), and `trace` in the `-e` option of the test start command (`mcftules`, `mcfauaps`, or `mcftusgs`).

Chapter

8. Test Execution

This chapter explains how to start and end a test, how duplicate test mode specifications are handled, and how to inherit and edit test mode information.

This chapter contains the following sections:

8.1 Starting and ending a test
8.2 Duplicate test mode specifications
8.3 Inheriting test mode information
8.4 Editing test information

8.1 Starting and ending a test

Test mode is the system status from execution of a test start command (`mcftules`, `mcfauaps`, or `mcftusgs`) until execution of a test end command (`mcftulee`, `mcfauape`, or `mcftusge`). The MCF online tester facilities can be used during this time.

8.1.1 Starting a test and setting the test environment

To use the MCF online tester, first enter the `mcfutfst` command to declare usage. Then, enter a test start command to start testing. Specify the *test environment* (the MCF online tester facilities to be used) in the test start command. These specifications are called *test mode information*.

Before starting a test, you can check whether the facilities of the MCF online tester are available. Enter the `mcflsutf` command to display tester status.

(1) Starting a test

(a) Testing a logical terminal

Enter the `mcftules` command to start testing. At command execution, the specified logical terminal is in test mode. That is, all the applications activated from the logical terminal run in test mode.

(b) Testing an application

Enter the `mcfauaps` command to start testing. At command execution, the specified application runs in test mode.

An application test can be performed when adding new application processing to an existing UAP.

(c) Testing a service group

Enter the `mcftusgs` command to start testing. At command execution, the specified service group enters test mode.

(d) Note on executing a test start command

Do not enter a test start command before shutdown of connection and completion of all message send and receive. If a test start command is executed during message send and receive, the application(s) run in test mode when subsequently activated.

(2) Setting the test environment

To set the test environment, specify any of the following facilities in the test start command:

- Disable updating of non-MCF resources

- Invalidate send messages
- Invalidate application startup messages
- Suppress error events
- Suppress MHP automatic shutdown
- Collecting UAP trace information

(3) Test mode range

The input messages for an application in test mode and the messages input from a logical terminal in test mode are called *test mode messages*.

Test mode is effective from the time the MHP receives a test mode message until the end of messages generated during testing.

8.1.2 Ending a test

To declare test termination, enter the test end command (`mcftulee`, `mcfauape` or `mcftusge`) from a workstation that accepts online tester operating commands.

When the test end command is executed, the specified logical terminal, application, or service group is released from test mode.

8.2 Duplicate test mode specifications

When two or more test mode specifications apply to an application, the precedence of the test environment specification for the application is in the order of first the logical terminal, then the application, and finally the service group.

For example, if an application is input from a logical terminal in test mode, and if a test environment is specified for that application by entering the `mcfauaps` command, the test environment specified for the logical terminal by the `mcftules` command takes effect. The test environment specified by the `mcfauaps` command is effective if the application is input from a logical terminal that is not in test mode.

Table 8-1 shows how duplicate test mode specifications for a logical terminal, an application, and a service group are handled.

Table 8-1: Duplicate test mode specifications

Logical terminal	Application	Service group	Source of valid test mode information
Y	Y	Y	Logical terminal
		N	Logical terminal
	N	Y	Logical terminal
		N	Logical terminal
N	Y	Y	Application
		N	Application
	N	Y	Service group
		N	--

Legend:

Y: The test mode is specified.

N: The test mode is not specified.

--: Not applicable.

8.3 Inheriting test mode information

When a test MHP issues the `dc_mcf_execap` function (for activating an application program), the test mode information specified in the test start command is inherited to the activated MHP.

Table 8-2 shows how test mode information is inherited.

Table 8-2: Inheritance of test mode information

<table>
<tr><th rowspan="4">Logical terminal</th><th rowspan="4">Application</th><th rowspan="4">Service group</th><th colspan="4">Application specified using dc_mcf_execap</th></tr>
<tr><th colspan="2">Y</th><th colspan="2">N</th></tr>
<tr><th colspan="2">Service group</th><th colspan="2">Service group</th></tr>
<tr><th>Y</th><th>N</th><th>Y</th><th>N</th></tr>
<tr><td rowspan="4">Y</td><td rowspan="2">Y</td><td>Y</td><td colspan="4" rowspan="4">The test mode information for the logical terminal is inherited.</td></tr>
<tr><td>N</td></tr>
<tr><td rowspan="2">N</td><td>Y</td></tr>
<tr><td>N</td></tr>
<tr><td rowspan="4">N</td><td rowspan="2">Y</td><td>Y</td><td colspan="2" rowspan="4">The test mode information for the application specified using <code>dc_mcf_execap</code> is inherited.</td><td rowspan="4">The test mode information for the MHP started using <code>dc_mcf_execap</code> is inherited.</td><td rowspan="2">The test mode information for the startup source application is inherited.</td></tr>
<tr><td>N</td></tr>
<tr><td rowspan="2">N</td><td>Y</td><td>The test mode information for the startup source MHP is inherited.</td></tr>
<tr><td>N</td><td>--</td></tr>
</table>

Legend:

Y: The test mode is specified.

N: The test mode is not specified.

--: Not applicable.

8.4 Editing test information

8.4.1 Displaying test mode information

Entering a test mode information display command (`mcftulsle`, `mcfaulsap`, or `mcftulssg`) can output the test mode information for a logical terminal, application, or service group (MHP) specified in a test start command to standard output. This facility allows the operator to monitor the status of the online test.

8.4.2 Collecting UAP trace information

To collect UAP (MHP) trace information, first complete the following specifications:

- Specify `uto_conf=Y` in the system service configuration definition.
- Specify the maximum size of the trace file in the `max_trace_file_size` operand of the tester service definition.
- Specify `test_mode=no` in the user service definition.
- Specify the test user ID in the `-u` option of the `mcfutfst` (MCF online tester use declaration) command.
- Specify `trace` in the `-e` option of the command for starting the MCF online tester.

8.4.3 Merging and outputting UAP trace information

MHP trace information is collected when these specifications are completed. The information can be edited and output to standard output by entering the online tester's `utotrcout` command.

The output format follows TP1/Server Base online tester specifications. No trace information is output for the following functions.

- `dc_mcf_open` function
- `dc_mcf_close` function
- `dc_mcf_mainloop` function
- `dc_mcf_regster` function

MHP trace information is output to the `$DCDIR/spool/uto/`*test-user-ID* directory. The test user ID in the pathname is the ID specified in the `-u` option of the `mcfutfst` command.

There are two MHP trace files, `trace1` and `trace2`. These files are swapped if the contents written to a file exceed the value specified in the `max_trace_file_size` operand of the tester service definition. A message reports that the files were swapped when one file became full. When this message is output, copy the contents of the full trace file to another file, then delete the full trace file.

See Section *3.1 System definitions for the online tester* for information on the system service configuration definition, tester service definition, and user service definition. For details on the `utotrcout` command, see Section *5.1 Operating commands for running tests*.

Chapter

9. Operating Commands

This chapter explains how to use the operating commands of the MCF online tester.

This chapter contains the following sections:

9.1 Operating commands for running tests
9.2 Operating commands for testing a logical terminal
9.3 Operating commands for testing an application
9.4 Operating commands for testing a service group

9.1 Operating commands for running tests

The following pages explain the operating commands for the MCF online tester. For information on command syntax and rules, see the manual *OpenTP1 Operation*.

Table 9-1 lists the operating commands for running tests.

Table 9-1: List of operating commands

Command name	Function
`mcfutfst`	MCF online tester use declaration
`mcflsutf`	Display of MCF online tester status

9.1.1 mcfutfst (MCF online tester use declaration)

(1) Syntax

```
mcfutfst [-u test-user-ID]
```

(2) Function

Declares usage of the MCF online tester.

Commands for the MCF online tester other than `mcfutfst` and `mcflsutf` are not accepted unless usage of the MCF online tester is first declared by entering the `mcfutfst` command.

The `mcfutfst` command is not accepted if usage of the MCF online tester has already been declared.

Enter the `mcfutfst` command only after shutdown of connection and completion of all message send and receive.

The MCF online tester ends when TP1/Message Control terminates.

(3) Option

- `-u` *test-user-ID* ~<identifier of 1-4 characters>

  Specify a test user ID for identifying the trace file directory.

  This option must be specified to collect MCF trace information.

9.1.2 mcflsutf (display of MCF online tester status)

(1) Syntax

```
mcflsutf
```

(2) Function

Outputs the status of the MCF online tester, showing whether the tester facilities can be used, to standard output.

Before the MCF online tester facilities can be used, usage must be declared by entering the `mcfutfst` command.

(3) Output format

```
A00   MCF mode=TEST   test user ID=mhp
1.              2.                3.
```

1. MCF manager process ID and MCF communication process ID
2. MCF mode indication
3. `TEST`

   MCF online tester can be used.

   `NORMAL`

   MCF online tester cannot be used.
4. `****` is displayed when no test user ID is specified.

9.2 Operating commands for testing a logical terminal

This section explains the commands of the MCF online tester for running a test on a logical terminal. For the format and rules of the operating commands, see the manual *OpenTP1 Operation*.

Table 9-2 lists the operating commands used for running tests on a logical terminal.

Table 9-2: Operating commands for running tests on a logical terminal

Command name	Function
`mcftulsle`	Display of test mode information for a logical terminal
`mcftules`	Start of a logical terminal test
`mcftulee`	Termination of a logical terminal test

9.2.1 mcftulsle (display of test mode information for a logical terminal)

(1) Syntax

`mcftulsle -l` *logical-terminal-name*

(2) Function

Outputs test mode information for the specified logical terminal to standard output.

(3) Option

- `-l` *logical-terminal-name* ~<identifier of 1-8 characters>

  Specify the name of the logical terminal for which test mode information is to be displayed.

  Specify an asterisk (*) in *logical-terminal-name* to display test mode information for all logical terminals in test mode. To display test mode information for all logical terminals whose names begin with a particular prefix character string, follow the prefix character string with an asterisk (*prefix-character-string*`*`).

  Only one logical terminal name can be specified.

(4) Output format

```
A01 LEual1 back trac swms erre exec hold
1.   2.    3.   4.   5.   6.   7.   8.
```

1. MCF manager process ID and MCF communication process ID

2. Logical terminal name (up to 8 characters)
3. Shows whether to restore resources to pre-test status at completion of a transaction.

   `back`

   Restore.

   `nobk`

   Do not restore.
4. Shows whether to collect MHP trace information during processing of a transaction in test mode.

   `trac`

   Collect.

   `notr`

   Do not collect.
5. Shows whether to invalidate MHP send messages issued by a transaction in test mode.

   `swms`

   Invalidate.

   `nosw`

   Do not invalidate.
6. Shows whether to suppress error event activation.

   `erre`

   Suppress.

   `noer`

   Do not suppress.
7. Shows whether to invalidate application startup messages issued by a transaction in test mode.

   `exec`

   Invalidate.

   `noex`

   Do not invalidate.
8. Shows whether to suppress MHP automatic shutdown should a transaction in test

mode terminate abnormally.

`hold`

Suppress.

`noho`

Do not suppress.

9.2.2 mcftules (start of a logical terminal test)

(1) Syntax

```
mcftules  [-e "[backout] [trace] [swmsg] [errevt]
          [execap] [holdlimit]"]
          -l logical-terminal-name
```

(2) Function

Sets the specified logical terminal in test mode and starts testing.

Enter the `mcftules` command only after shutdown of connection and completion of all message send and receive.

(3) Options

- `-e`

Specify the test mode options.

Enclose two or more flag arguments with quotation marks (") and delimit each flag argument by inserting a space.

Flag arguments:

ba`ckout`

Restores the resources used in a transaction to pre-test status at completion of the transaction.

When this flag argument is omitted, updated resources are used in their current status and are not restored to pre-test status.

tr`ace`

Collects MHP trace information during processing of a transaction in test mode.

When this flag argument is omitted, no MHP trace information is collected.

sw`msg`

Invalidates messages sent by the MHP during processing of a transaction in test mode.

Messages sent by the following functions issued by the test MHP are invalidated:

`dc_mcf_send` function (message send)

`dc_mcf_sendsync` function (synchronous message send)

`dc_mcf_resend` function (message resend)

When this flag argument is omitted, messages sent by the above functions are effective.

er`revt`

Suppresses error event activation if an error event occurs during testing. The following error events are suppressed:

`ERREVT1` (MCF event that reports detection of an invalid application name)

`ERREVT2` (MCF event that reports discarding of a message at abnormal termination before issue of the `dc_mcf_receive` function)

`ERREVT2` (MCF event that reports discarding of a message generated at automatic shutdown)

`ERREVT3` (MCF event that reports UAP abnormal termination at abnormal termination during MHP execution)

When this flag argument is omitted, activation of the above error events is not suppressed.

ex`ecap`

Invalidates branch application startup messages issued by a transaction in test mode.

When this flag argument is omitted, branch application startup messages are effective.

ho`ldlimit`

Suppresses MHP automatic shutdown should a transaction in test mode terminate abnormally.

When this flag argument is omitted, MHP automatic shutdown is not suppressed.

- `-l` *logical-terminal-name* ~<identifier of 1-8 characters>

Specify the name of the logical terminal at which to start testing.

You cannot use an asterisk (*) or a prefix character string plus an asterisk (prefix-character-string*) to specify a group of logical terminal names.

Only one logical terminal name can be specified.

9.2.3 mcftulee (termination of a logical terminal test)

(1) Syntax

```
mcftulee  -l logical-terminal-name
```

(2) Function

Releases test mode status at the specified logical terminal and ends testing.

(3) Option

- `-l` *logical-terminal-name*~<identifier of 1-8 characters>

  Specify the name of the logical terminal at which to end testing.

  You cannot use an asterisk (*) or a prefix character string plus an asterisk (prefix-character-string*) to specify logical terminal names in a batch.

  Only one logical terminal name can be specified.

9.3 Operating commands for testing an application

This section explains the commands of the MCF online tester for running a test on an application. For the format and rules of the operating commands, see the manual *OpenTP1 Operation*.

Table 9-3 lists the operating commands used for running tests on an application.

Table 9-3: Operating commands for running tests on an application

Command name	Function
`mcfaulsap`	Display of test mode information for an application
`mcfauaps`	Start of an application test
`mcfauape`	Termination of an application test

9.3.1 mcfaulsap (display of test mode information for an application)

(1) Syntax

```
mcfaulsap  -s { MCF-communication-process-ID |
                application-startup-process-ID }
           -a application-name [-k application-name-type]
```

(2) Function

Outputs test mode information for the specified application to standard output.

(3) Options

- `-s` *MCF-communication-process-ID* |

  application-startup-process-ID ~<hexadecimal> ((01-ef))

  Specify the MCF communication process ID or application startup process ID.

  Specify the application startup process ID when testing an application specified by ERREVT or by the `dc_mcf_execap` function. In all other cases, specify the MCF communication process ID.

  Only one process ID can be specified.

- `-a` *application-name* ~<identifier of 1-8 characters>

  Specify the name of the application for which test mode information is to be displayed.

  Specify an asterisk (*) in *application-name* to display test mode information for all applications in test mode. Placing an asterisk (*) after first character(s) of the application name (first_characters_*) shows test mode information for all applications whose name begins with those character before *.

Only one application name can be specified.

- `-k` *application-name-type*

  Specify the type of the application specified in the `-a` option:

  `user`

  User application

  `mcf`

  MCF event

  When this option is omitted, the application name specified in the `-a` option is assumed to be a user application name.

(4) Output format

```
A01 user aprep01 back trac swms erre exec hold
 1.  2.     3.    4.   5.   6.   7.   8.   9.
```

1. MCF manager process ID, MCF communication process ID, or application startup process ID
2. Application name type

   `user`

   User application

   `mcf`

   MCF event
3. Application name or MCF event name
4. Shows whether to restore resources to pre-test status at completion of a transaction.

   `back`

   Restore.

   `nobk`

   Do not restore.
5. Shows whether to collect MHP trace information during processing of a transaction in test mode.

   `trac`

Collect.

`notr`

Do not collect.

6. Shows whether to invalidate MHP send messages issued by a transaction in test mode.

   `swms`

   Invalidate.

   `nosw`

   Do not invalidate.

7. Shows whether to suppress error event activation.

   `erre`

   Suppress.

   `noer`

   Do not suppress.

8. Shows whether to invalidate application startup messages issued by a transaction in test mode.

   `exec`

   Invalidate.

   `noex`

   Do not invalidate.

9. Shows whether to suppress MHP automatic shutdown should a transaction in test mode terminate abnormally.

   `hold`

   Suppress.

   `noho`

   Do not suppress.

9.3.2 mcfauaps (start of an application test)

(1) Syntax

```
mcfauaps  -s { MCF-communication-process-ID |
               application-startup-process-ID }
         [-e "[backout] [trace] [swmsg] [errevt]
         [execap] [holdlimit]"]
          -a application-name [-k application-name-type]
```

(2) Function

Sets the specified application in test mode and starts testing.

Enter the `mcfauaps` command only after shutdown of connection and completion of all message send and receive.

(3) Options

- `-s` *MCF-communication-process-ID* |

  application-startup-process-ID ~<hexadecimal> ((01-ef))

  Specify the MCF communication process ID or application startup process ID.

  To test an application specified by the `dc_mcf_execap` function, specify the application startup process ID. To test ERREVT, specify IDs based on Table 9-4. For other testing, specify the MCF communication process ID.

  Only one process ID can be specified.

  Table 9-4: IDs to be specified when testing ERREVT (mcfauaps command)

ERREVT to be tested	ID to be specified	
	MCF communication process ID	Application startup process ID
Invalid application name notification event (ERREVT1)	Y	--
Message discard event that is issued by abnormal termination before the `dc_mcf_receive` function is issued (ERREVT2)	--	Y
Message discard event generated by shutdown (ERREVT2)	Y#	Y#
UAP abnormal termination notification event that is issued by abnormal termination during MHP execution (ERREVT3)	--	Y

Legend:

Y: Specifiable.

--: Not specifiable.

#: Specify both the MCF communication process ID and the application startup process ID.

■ `-e`

Specify the test mode options.

Enclose two or more flag arguments with quotation marks (") and delimit each flag argument by inserting a space.

Flag arguments:

backout

Restores the resources used in a transaction to pre-test status at completion of the transaction.

When this flag argument is omitted, updated resources are used in their current status and are not restored to pre-test status.

trace

Collects MHP trace information during processing of a transaction in test mode.

When this flag argument is omitted, no MHP trace information is collected.

swmsg

Invalidates messages sent by the MHP during processing of a transaction in test mode.

Messages sent by the following functions issued by the test MHP are invalidated:

`dc_mcf_send` function (message send)

`dc_mcf_sendsync` function (synchronous message send)

`dc_mcf_resend` function (message resend)

When this flag argument is omitted, messages sent by the above functions are effective.

errevt

Suppresses error event activation if an error event occurs during testing. The following error events are suppressed:

`ERREVT1` (MCF event that reports detection of an invalid application name)

However, the above error event can be suppressed only when testing is performed in a logical terminal.

`ERREVT2` (MCF event that reports discarding of a message at abnormal termination before issue of the `dc_mcf_receive` function)

`ERREVT2` (MCF event that reports discarding of a message generated at automatic shutdown)

`ERREVT3` (MCF event that reports UAP abnormal termination at abnormal termination during MHP execution)

When this flag argument is omitted, activation of the above error events is not suppressed.

`ex``ecap`

Invalidates branch application startup messages issued by a transaction in test mode.

When this flag argument is omitted, branch application startup messages are effective.

`ho``ldlimit`

Suppresses MHP automatic shutdown should a transaction in test mode terminate abnormally.

When this flag argument is omitted, MHP automatic shutdown is not suppressed.

- `-a` *application-name* ~<identifier of 1-8 characters>

Specify the name of the application to be tested.

You cannot use an asterisk (*) or a prefix character string plus an asterisk (prefix-character-string*) to specify application names in a batch.

Only one application name can be specified.

- `-k` *application-name-typet*

Specify the type of the application specified in the `-a` option:

`user`

User application

`mcf`

MCF event

When this option is omitted, the application name specified in the `-a` option is assumed to be a user application name.

9.3.3 mcfauape (termination of an application test)

(1) Syntax

```
mcfauape  -s { MCF-communication-process-ID |
               application-startup-process-ID }
          -a application-name [-k application-name-type]
```

(2) Function

Releases test mode at the specified application and ends testing.

(3) Options

- `-s` *MCF-communication-process-ID* |

  application-startup-process-ID ~<hexadecimal> ((01-ef))

Specify the MCF communication process ID or application startup process ID.

Specify the application startup process ID when testing an application specified using the `dc_mcf_execap` function. When testing ERREVT, specify the applicable ID as shown in Table 9-5. In all other cases, specify the MCF communication process ID.

Only one process ID can be specified.

Table 9-5: IDs to be specified when testing ERREVT (mcfauape command)

ERREVT to be tested	ID to be specified	
	MCF communication process ID	Application startup process ID
Invalid application name notification event (ERREVT1)	Y	--
Message discard event that is issued by abnormal termination before the `dc_mcf_receive` function is issued (ERREVT2)	--	Y
Message discard event generated by shutdown (ERREVT2)	Y#1	Y#2
UAP abnormal termination notification event that is issued by abnormal termination during MHP execution (ERREVT3)	--	Y

Legend:

Y: Specifiable.

--: Not specifiable.

#1: Specify when the application activated by the received message is shut down.

#2: Specify when the application activated by the `dc_mcf_execap` function is shut down.

- `-a` *application-name* ~<identifier of 1-8 characters>

  Specify the name of the application at which to end testing.

  You cannot use an asterisk (*) or a prefix character string plus an asterisk (prefix-character-string*) to specify application names in a batch.

  Only one application name can be specified.

- `-k` *application-name-type*

  Specify the type of the application specified in the `-a` option:

  `user`

  User application

  `mcf`

  MCF event

  When this option is omitted, the application name specified in the `-a` option is assumed to be a user application name.

9.4 Operating commands for testing a service group

This section explains the commands of the MCF online tester for running a test on a service group. For the format and rules of the operating commands, see the manual *OpenTP1 Operation*.

Table 9-6 lists the operating commands used for running tests on a service group.

Table 9-6: Operating commands for running tests on an application

Command name	Function
`mcftulssg`	Display of test mode information for a service group
`mcftusgs`	Start of a service group test
`mcftusge`	Termination of a service group test

9.4.1 mcftulssg (display of test mode information for a service group)

(1) Syntax

`mcftulssg -g` *service-group-name*

(2) Function

Outputs test mode information for the specified service group to standard output.

(3) Option

- `-g` *service-group-name* ~<identifier of 1-31 characters>

  Specify the service group name.

  Specifying an asterisk (*) for the service group name outputs test mode information for all the service groups in test mode. Placing an * after first character(s) of the service group name (first_characters_*) shows test mode information for all service groups whose name begins with those character before *.

(4) Output format

```
A01 SVG01 LEual1 back trac swms erre exec hold
1.  2.    3.     4.   5.   6.   7.   8.   9.
```

1. MCF manager process ID and MCF communication process ID
2. Service group name

3. Logical terminal name (no more than 8 characters)
4. Shows whether to restore the resource to the status before the test when the transaction ends.

   `back`

   Restore.

   `nobk`

   Do not restore.
5. Shows whether to collect the trace information of the MHP while a test mode transaction is being processed.

   `trac`

   Collect.

   `notr`

   Do not collect.
6. Shows whether to invalidate the send message issued by a test mode transaction.

   `swms`

   Invalidate.

   `nosw`

   Do not invalidate.
7. Shows whether to suppress the startup of error events.

   `erre`

   Suppress.

   `noer`

   Do not suppress.
8. Shows whether to invalidate the application startup message issued by a test mode transaction.

   `exec`

   Invalidate.

   `noex`

   Do not invalidate.
9. Shows whether to suppress the automatic shutdown function of the MHP if a test mode transaction ends abnormally.

`hold`

Suppress.

`noho`

Do not suppress.

9.4.2 mcftusgs (start of a service group test)

(1) Syntax

```
mcftusgs -g service-group-name
        [-e"[backout] [trace] [swmsg] [errevt]
            [execap] [holdlimit]"]
```

(2) Function

Sets the specified service group in test mode. The `mcftusgs` command must be executed after the connection is shut down and there is no transmission of messages.

(3) Options

- `-g` *service-group-name* ~<identifier of 1-31 characters>

  Specify the name of the service group where the test is to be started.

  You cannot use an asterisk (*) or a prefix character string plus an asterisk (prefix-character-string*) to specify a group of service group names.

  Only one service group name can be specified.

- `-e`

  Specify the test mode options.

  Enclose two or more flag arguments with quotation marks (") and delimit each flag argument by inserting a space.

  Flag arguments:

  ba`ckout`

  Restores the resource used in a transaction to pre-test status at completion of the transaction.

  When this flag argument is omitted, updated resources are used in their current status and are not restored to pre-test status.

  tr`ace`

  Collects MHP trace information during processing of a transaction in test mode.

  When this flag argument is omitted, no MHP trace information is collected.

swmsg

Invalidates messages sent by MHP during processing of a transaction in test mode. Messages sent by the following functions issued by the test MHP are invalidated:

dc_mcf_send function (message send)

dc_mcf_sendsync function (synchronous message send)

dc_mcf_resend function (message resend)

When this flag argument is omitted, messages sent by the above functions are effective.

errevt

Suppresses error event activation if an error event occurs during testing. The following error events are suppressed:

ERREVT1 (MCF event that reports discarding of a message at abnormal termination before issue of the dc_mcf_receive function)

However, the above error event can be suppressed only when testing is performed in a logical terminal.

ERREVT2 (MCF event that reports discarding of a message at abnormal termination before issue of the dc_mcf_receive function)

ERREVT2 (MCF event that reports discarding of a message generated at automatic shutdown)

ERREVT3 (MCF event that reports UAP abnormal termination at abnormal termination during MHP execution)

When this flag argument is omitted, activation of the above error events is not suppressed.

execap

Invalidates the branch application startup message issued by a transaction in test mode.

When this flag argument is omitted, branch application startup messages are effective.

holdlimit

Suppresses MHP automatic shutdown should a transaction in test mode terminates.

When this flag argument is omitted, MHP automatic shutdown is not suppressed.

9.4.3 mcftusge (termination of a service group test)

(1) Syntax

```
mcftusge -g service-group-name
```

(2) Function

Releases test mode for the specified service group and ends the test.

(3) Option

- `-g` *service-group-name* ~<identifier of 1-31 characters>

  Specify the service group name for which the test should terminate.

  You cannot use an asterisk (*) or a prefix character string plus an asterisk (prefix-character-string*) to specify service group names in a batch.

  Only one service group name can be specified.

Chapter

10. Facilities

This chapter describes the test facilities available with the offline tester.

This chapter contains the following sections:

10.1 Facilities of the offline tester
10.2 Simulating a client UAP
10.3 Simulating a server UAP
10.4 Simulating the MCF
10.5 Simulating file services
10.6 Simulating OpenTP1 functions
10.7 Simulating operating commands
10.8 Creating tester files
10.9 Continuous command execution
10.10 Debugger connection
10.11 Collecting test information

10.1 Facilities of the offline tester

The offline tester provides the following facilities for testing UAPs:

1. Client UAP simulator

   Simulates client UAP processing so that a server UAP can be tested without a client UAP.

2. Server UAP simulator

   Simulates server UAP processing so that a client UAP can be tested without a server UAP.

3. MCF simulator

   Simulates message send and receive processing for testing an MHP or an SPP called by service requests from the MHP.

4. File service simulators

   Simulate the DAM service and TAM service for testing UAP access to DAM or TAM files.

5. OpenTP1 function simulator

   Simulates processing of OpenTP1 functions by using the corresponding simulation functions that have the same names as the OpenTP1 functions.

6. Operating command simulator

   Simulates the processing of operating commands executed by a test UAP.

7. Tester file creation

   Creates the tester files required when using the simulators.

8. Continuous command execution

   During testing, continuously executes the offline tester subcommands set in a file.

9. Debugger connection

   Runs test UAPs under debugger control.

10. Collection of offline tester trace information

    Collects trace information for the UAP being tested.

10.2 Simulating a client UAP

The offline tester can take the place of a client UAP in requesting services from a server UAP. This allows the user to test the server UAP without needing a client UAP. This facility is called the *client UAP simulator*.

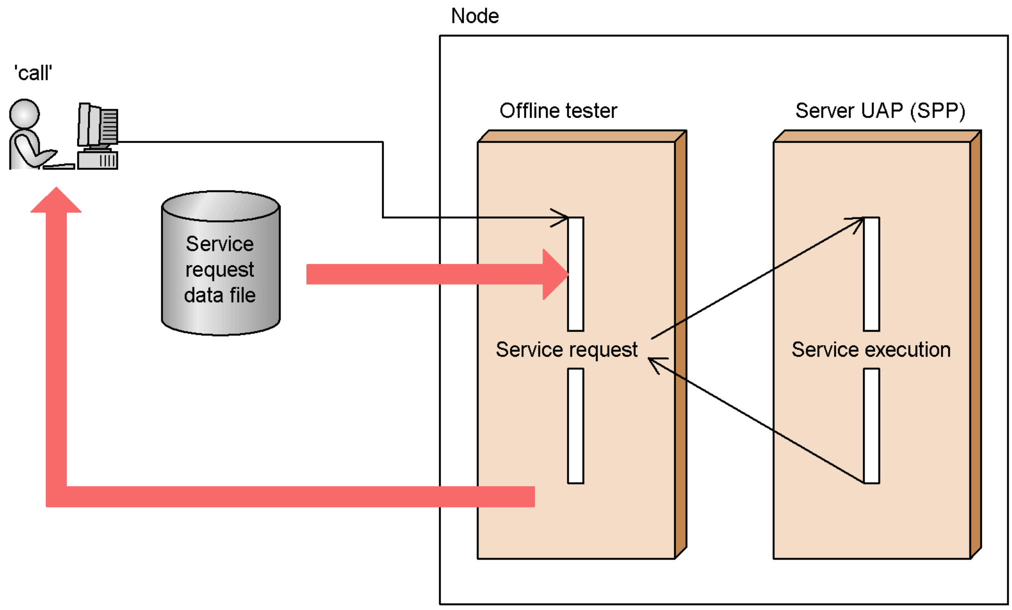
An offline tester command is used to simulate a client UAP. Before executing the command, the user must first create the processing data to be passed to the server UAP. This data is created in a *service request data file*.

There are three types of service request data files which are used according to the client interface:

- RPC request data file (for simulating a UAP that has an RPC interface)
- XATMI request data file (for simulating a UAP that has an XATMI interface)
- TxRPC request data file (for simulating a UAP that has a TxRPC interface)

Figure 10-1 outlines the client UAP simulator.

Figure 10-1: Simulating a client UAP



10.2.1 Simulating a client UAP with an RPC interface

To simulate a client UAP that uses an RPC interface to send service requests, the user must first create an RPC request data file containing the processing data to be passed to the server UAP being tested.

10.2.2 Simulating a client UAP with an XATMI interface

To simulate a client UAP that uses an XATMI interface to send service requests, the user must first create an *XATMI request data file* containing the processing data to be passed to the server UAP being tested.

When service requests are made interactively, the user must also create an *XATMI receive data file* containing the test data to be received by the server UAP during service execution. If the server UAP passes send data, the offline tester makes a file name inquiry for each service. Using an offline tester command, the user specifies the name of an *XATMI send data file* for saving the send data.

10.2.3 Simulating a client UAP with a TxRPC interface

To simulate a client UAP that uses a TxRPC interface to send service requests, the user must first create a *TxRPC request data file* containing the processing data to be passed to the server UAP being tested.

10.3 Simulating a server UAP

The offline tester can take the place of a server UAP in executing services requested by a client UAP. This allows the user to test the client UAP without needing a server UAP. This facility is called the *server UAP simulator*.

To simulate a server UAP, the user activates the server UAP (dummy) and then executes an OpenTP1 command. Before executing the command, the user must create the response data to be passed to the client UAP. This data is created in a *service response data file*. When the client UAP sends a service request, the offline tester reads the response data from the file and passes it to the client UAP.

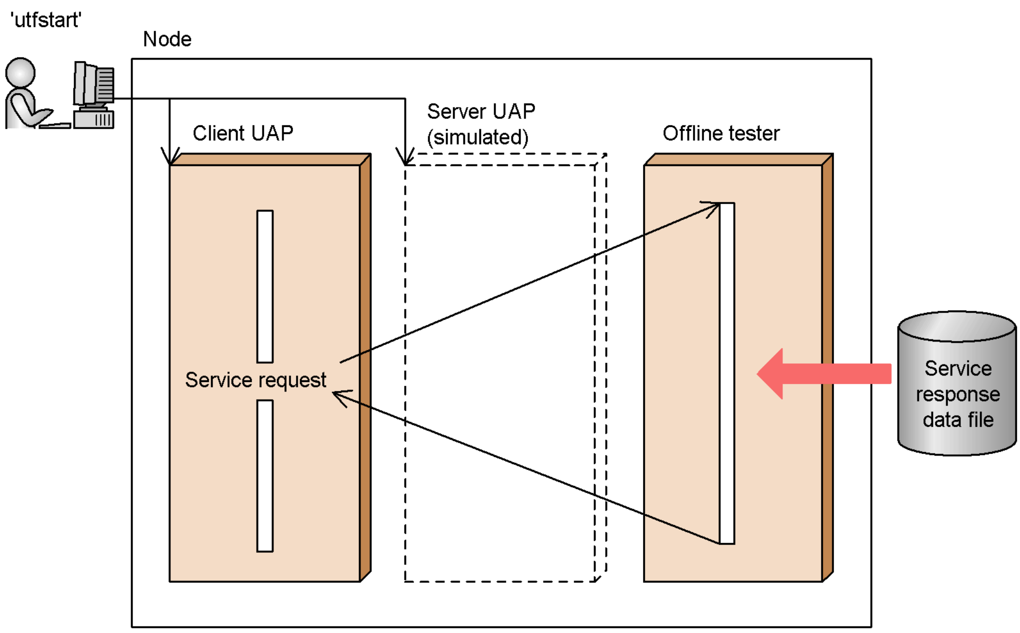
There are three types of service response data files which are used according to the UAP interface:

- RPC response data file (for simulating a UAP that has an RPC interface)
- XATMI response data file (for simulating a UAP that has an XATMI interface)
- TxRPC response data file (for simulating a UAP that has a TxRPC interface)

To simulate a server UAP, the user must first define the server UAP as the *simulation target* in an offline tester environment definition. This enables the server UAP to be simulated but not actually activated when a test is performed.

Figure 10-2 outlines the server UAP simulator.

Figure 10-2: Simulating a server UAP



10.3.1 Simulating a server UAP with an RPC interface

To simulate a server UAP that uses an RPC interface for accepting service requests, the user must first create an *RPC response data file* with the response data to be returned to the client UAP. When the client UAP sends a service request, the offline tester reads the response data from the file and returns it to the client UAP.

10.3.2 Simulating a server UAP with an XATMI interface

To simulate a server UAP that uses an XATMI interface for accepting service requests, the user must first create an *XATMI response data file* with the response data to be returned to the client UAP. When the client UAP sends a service request, the offline tester reads the response data from the file and returns it to the client UAP.

When service requests are made interactively, the user must also create an *XATMI receive data file* containing the test data to be received by the client UAP during service execution. If the client UAP passes send data, the offline tester makes a file name inquiry for each service. Using an offline tester command, the user specifies the name of an *XATMI send data file* for saving the send data.

10.3.3 Simulating a server UAP with a TxRPC interface

To simulate a server that uses a TxRPC interface for accepting service requests, the user must first create a *TxRPC response data file* with the response data to be returned to the client UAP. When the client UAP sends a service request, the offline tester reads the response data from the file and returns it to the client UAP.

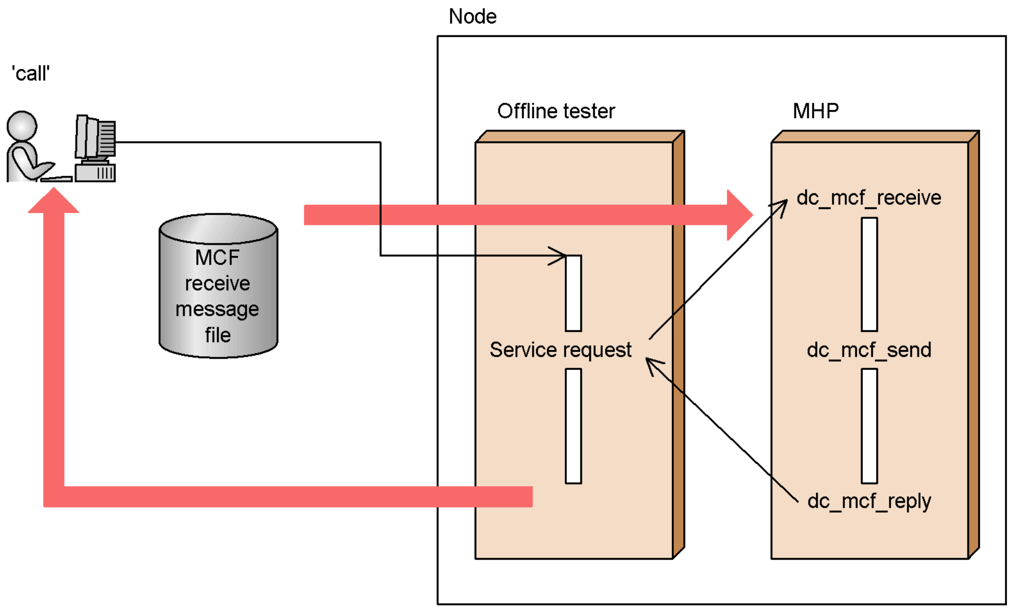
10.4 Simulating the MCF

The offline tester can take the place of the MCF in exchanging messages with an MHP. This allows the user to test the MHP without an MCF. This facility is called the *MCF simulator*.

An offline tester command is used to start the MHP application. Before executing the command, the user must first create an MCF receive message file with the messages to be passed to the MHP.

Figure 10-3 outlines the MCF simulator.

Figure 10-3: Simulating an MCF



10.5 Simulating file services

This section describes how the offline tester simulates file services in order to test file access.

10.5.1 Simulating the DAM service

The offline tester can simulate the DAM service for testing UAP access to DAM files. This facility is called the *DAM service simulator*.

Files created by an editor or by the function for simulating DAM file creation (`dc_dam_create` function) are handled by the TP1/FS/Direct Access file interface. The user must write an offline tester environment definition to associate each logical file name with the actual file.

At each update request from the UAP, a DAM file simulated by the offline tester is immediately updated (but writing is delayed). If the UAP terminates abnormally or if a rollback request occurs, the DAM file remains in updated status.

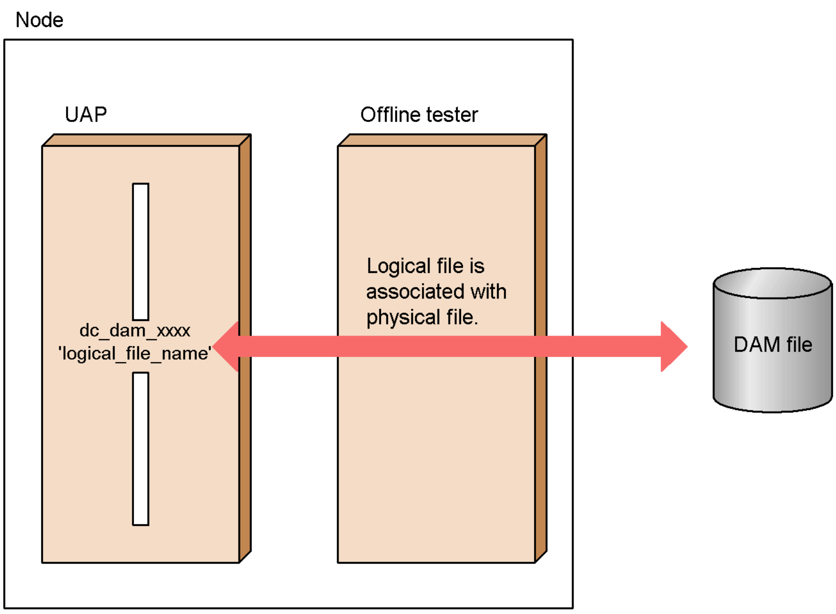
File update can be suppressed by option specification when starting the offline tester. Thus, the contents of a DAM file remain unchanged even if the UAP issues an update request function. If the data is re-entered after the update request, the file contents are the same as before the update request.

The user can also specify in the offline tester environment definition whether a lock is to be used for DAM files. Locks can only be placed on files, regardless of any specification made in a function.

Note that the DAM file is not closed when the `dc_trn_unchained_commit` function is issued in a UAP.

Figure 10-4 outlines the DAM service simulator.

Figure 10-4: Simulating the DAM service



10.5.2 Simulating the TAM service

The offline tester can simulate the TAM service for testing UAP access to TAM files. This facility is called the *TAM service simulator*.

Files created by the offline tester `utftamcre` command are handled by the TP1/FS/Table Access file interface. The user must write an offline tester environment definition to associate each logical file name with the actual file.

A TAM file simulated by the offline tester can access the same TAM data files as TP1/FS/Table Access. Indexing is also the same. However, TAM files cannot be accessed by DAM service functions. Also, to reduce shared memory size, only the management and index parts of the TAM file are stored in shared memory and the data part is accessed directly in the TAM file.

At each update request from the UAP, a TAM file simulated by the offline tester is immediately updated (but writing is delayed). If the UAP terminates abnormally or if a rollback request occurs, the TAM file remains in updated status.

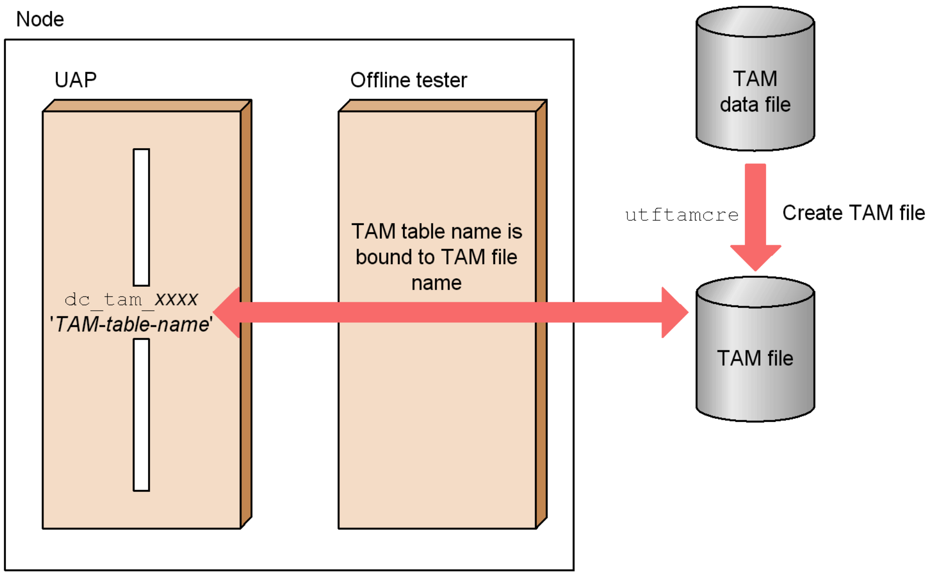
File update can be suppressed by option specification when starting the offline tester. Thus, the contents of a TAM file remain unchanged even if the UAP issues an update request function. If the data is re-entered after the update request, the file contents are the same as before the update request.

The user can also specify in the offline tester environment definition whether a lock is to be used for TAM files. Locks can only be placed on files, regardless of any specification made in a function.

Note that the TAM file is not closed when the `dc_trn_unchained_commit` function is issued in a UAP.

Figure 10-5 outlines the TAM service simulator.

Figure 10-5: Simulating the TAM service



10.6 Simulating OpenTP1 functions

The offline tester provides *simulation functions* which replace and have the same names as the functions provided by TP1/Server Base. These simulation functions can be used by linkage with the UAP.

The user can set the return values of the OpenTP1 functions in a *function return value file*. This facility enables set information to be returned to the UAP at completion of a simulation function. The facility operates if no error is detected when the offline tester performs the argument check. If an error is detected, the return code for the error is returned to the UAP.

For DAM and TAM-related functions, error return values set by the user are returned to the UAP. When a simulation function completes normally, however, the actual processing result is returned, not the set return value. That is, a return value set by the user is returned only if an error occurs during test processing.

For the `tpsend` and `tprecv` functions which use the XATMI interface, event names can be set in the function return value file. For TP1/Multi functions (function names beginning with `dc_adm_get_xxx`), the user can also set the output data (node ID and server name).

See Chapter *14. Simulation Functions* in this part of the manual for details on the return values that can be set for each simulation function.

When using a function provided by TP1/Shared Table Access, be sure to specify an IST table used by the function in the offline tester environment definition at offline tester startup.

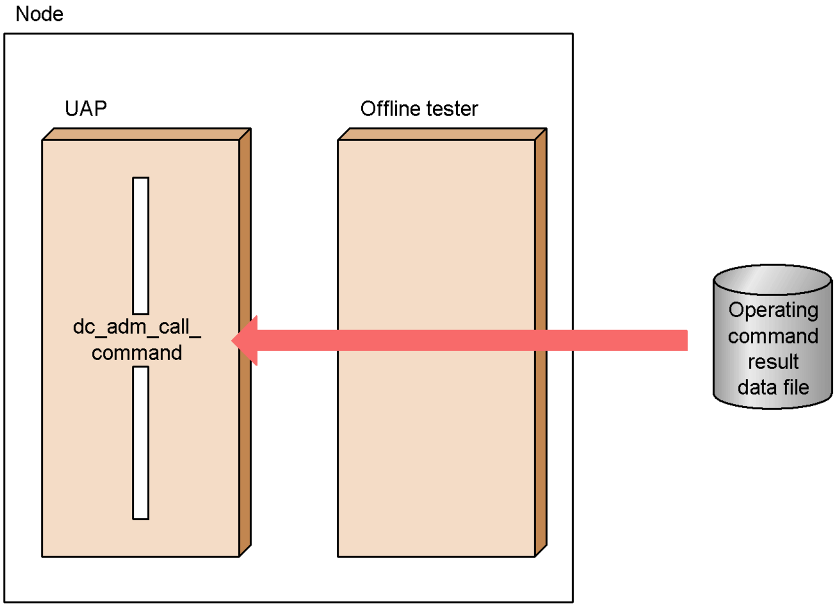
10.7 Simulating operating commands

The offline tester can simulate command execution requested by the `dc_adm_call_command` function issued in a UAP. This facility is called the *operating command simulator*.

To simulate operating command execution, the user must first create the execution result data in an *operating command result data file*. Then, when the `dc_adm_call_command` function is issued in the UAP, the offline tester reads the execution result data from the file and returns the data to the UAP.

Figure 10-6 outlines the operating command simulator.

Figure 10-6: Simulating UAP operating commands



10.8 Creating tester files

A data file must be created for each simulator provided by the offline tester. These are called *tester files*.

Each tester file is written in a specific data format. However, the user can easily create the tester files by command input, using the offline tester. This is called the *tester file creation facility*.

Table 10-1 lists the tester files that can be created using the tester file creation facility.

Table 10-1: Tester files created by tester file creation facility

Tester files		**Simulator using the tester file**
Service request data files	RPC request data file	Client UAP simulator
	XATMI request data file	Client UAP simulator
	TxRPC request data file	Client UAP simulator
Service response data files	RPC response data file	Server UAP simulator
	XATMI response data file	Server UAP simulator
	TxRPC response data file	Server UAP simulator
XATMI receive data file		Client UAP simulator
MCF receive message file		MCF simulator
Operating command result data file		Operating command simulator

To generate the tester files, the tester file creation facility uses data in a test data definition file that is created beforehand by the user. Use a text editor to create the data. Data for a number of tester files can be set in the same test data definition file.

Tester files can also be created in the required file format using a binary editor.

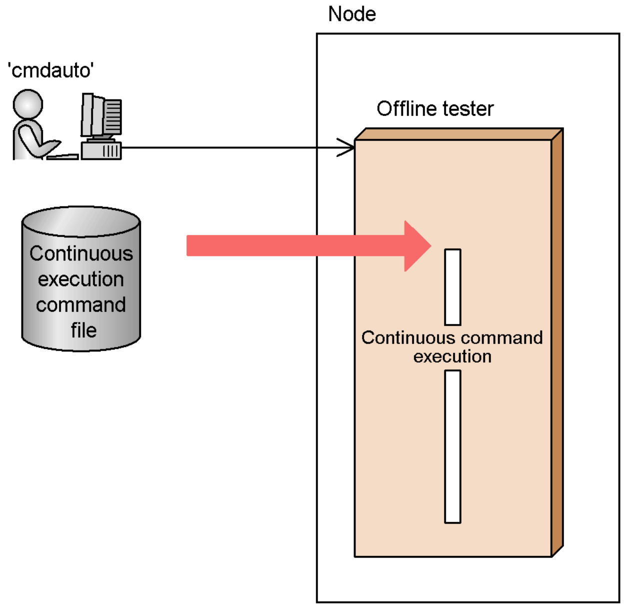
10.9 Continuous command execution

Offline tester commands can be set in a file for automatic sequential execution. This facility is called *continuous command execution*.

The commands to be executed are set in a *continuous execution command file*. The offline tester reads the file and executes the commands in the set sequence. Subcommands for responses are also executed if set. If no response subcommand is set in the file, the offline tester waits for user response. Thus, continuous command execution is useful when the testing sequence is fixed.

Figure 10-7 outlines continuous command execution.

Figure 10-7: Continuous command execution



10.10 Debugger connection

Using the offline tester, a UAP can be executed under debugger control from the `main` function. This facility is called *debugger connection*.

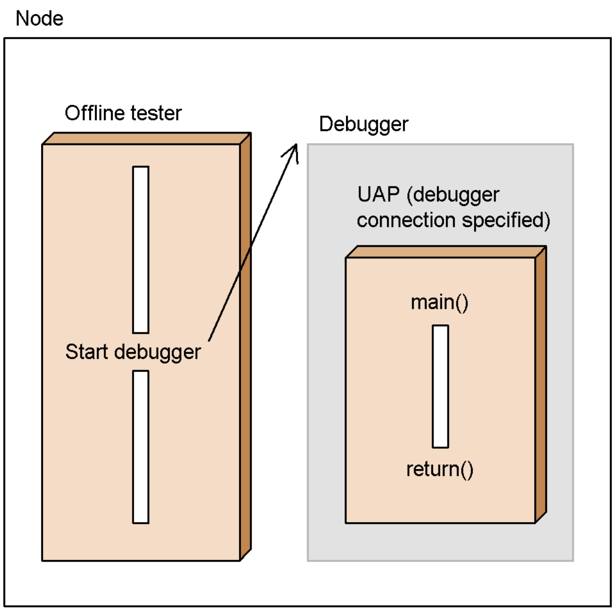
The user sets debugger connection in the offline tester environment definition. This makes it easy to debug each step of the program or to debug in batch format.

Two types of debuggers can be used:

- `dbx`
- `cbltd` (COBOL85/TD)

Figure 10-8 outlines debugger connection.

Figure 10-8: Debugger connection



10.11 Collecting test information

10.11.1 Collecting offline tester trace information

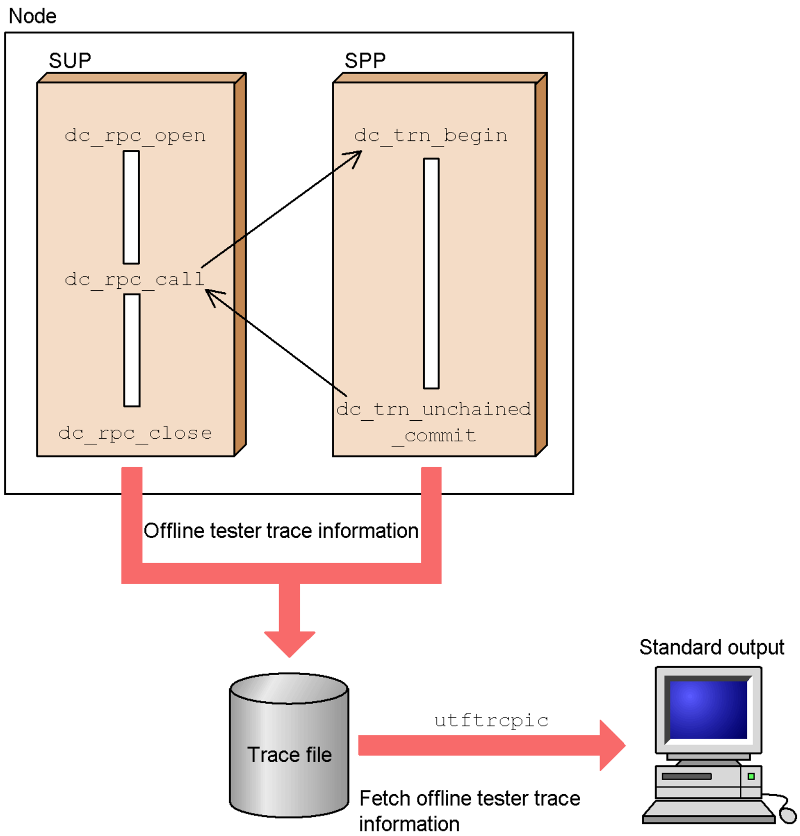
The offline tester can output the arguments and return information of OpenTP1 functions as trace information. This is called *collecting offline tester trace information*. The trace information can be output to standard output or to a file specified in the offline tester environment definition. Regardless of the output file, the output trace information is the same and has the same format.

Using an offline tester command, the user can also output information from a trace file for a selected service.

Do not use a shared trace file. The contents of the trace file are overwritten when a number of offline testers share the same file.

Figure 10-9 illustrates collection of offline tester trace information.

Figure 10-9: Collecting offline tester trace information



Chapter

11. Setting the Test Environment

This chapter explains how to set the environment for running tests with the offline tester.

This chapter contains the following sections:

11.1 System definitions for the offline tester
11.2 User-created files
11.3 Creating files

11.1 System definitions for the offline tester

The system definitions for running the offline tester are described below. See the manual *OpenTP1 System Definition* for information on definition structure and rules.

11.1.1 Offline tester environment definition

The offline tester environment definition specifies the following conditions for using the offline tester:

- UAP definition
- Directory definition for the RPC request data file
- Directory definition for the XATMI request data file
- Directory definition for the TxRPC request data file
- Directory definition for the RPC response data file
- Directory definition for the XATMI response data file
- Directory definition for the TxRPC response data file
- Directory definition for the XATMI send/receive data file
- Directory definition for the MCF receive message file
- Directory definition for the operating command result data file
- Directory definition for the continuous execution command file
- DAM file definitions
- TAM file definitions
- Internode shared table definitions
- Definition of the function return values file
- Trace file definition
- Protocol definition

Code each definition in the *offline tester environment definition file*. The file name is used as the command argument in the offline tester start command and can be any name.

Rules for the offline tester environment definition:

1. Write one definition per line.
2. Use one-byte characters. The system distinguishes between upper-case and lower-case characters.

3. End each line with a comma (,). Any coding after the comma is regarded as a comment.
4. End the whole environment definition with a semicolon (;). Any coding after the semicolon is regarded as a comment.
5. In the following cases, an error message is output at definition analysis. The definition is ignored and analysis continues:
    - When a non-existent directory and file (other than DAM file) is specified
    - When access to the specified file is prohibited (no write permission, for example)
    - When an error occurs during definition analysis

    When definition analysis is completed, the user must specify whether to continue offline tester activation. Enter either of the following:

    `1`

    To continue

    `2` (or `end`)

    To cancel
6. Do not abbreviate definitions. A format error occurs when a definition is abbreviated.
7. Table 11-1 shows whether each definition is valid or invalid if a format error occurs.

Table 11-1: Format errors and validity of definitions

Definition statement	Format error	Valid	Assumed specification when valid
UAP definition	No service group name, execution format program name, or user service definition file name	N	--
	Both `N` and `F` specified.	Y	`F`
	`,` or `;` missing at the end of the statement.	Y	`,`
Directory for RPC request data file	No directory name	N	--
	`,` or `;` missing at the end of the statement.	Y	`,`

Definition statement	Format error	Valid	Assumed specification when valid
Directory for XATMI request data file	No directory name	N	--
	, or ; missing at the end of the statement.	Y	,
Directory for TxRPC request data file	No directory name	N	--
	, or ; missing at the end of the statement.	Y	,
Directory for RPC response data file	No directory name	N	--
	, or ; missing at the end of the statement.	Y	,
Directory for XATMI response data file	No directory name	N	--
	, or ; missing at the end of the statement.	Y	,
Directory for TxRPC response data file	No directory name	N	--
	, or ; missing at the end of the statement.	Y	,
Directory for XATMI send/receive data file	No directory name	N	--
	, or ; missing at the end of the statement.	Y	,
Directory for MCF receive message file	No directory name	N	--
	, or ; missing at the end of the statement.	Y	,
Directory for operating command result data file	No directory name	N	--
	, or ; missing at the end of the statement.	Y	,
Directory for continuous execution command file	No directory name	N	--
	, or ; missing at the end of the statement.	Y	,
DAM file definition	No physical file name or logical file name.	N	--
	, or ; missing at the end of the statement.	Y	,

Definition statement	Format error	Valid	Assumed specification when valid
TAM file definition	No TAM table name or TAM file name	N	--
	`,` or `;` missing at the end of the statement.	Y	`,`
Internode shared table definition	No internode shared table name, record length, or number of records.	N	--
	`,` or `;` missing at the end of the statement.	Y	`,`
Definition of function return values file	No file name	N	--
	`,` or `;` missing at the end of the statement.	Y	`,`
Trace file definition	No file name	N	--
	`,` or `;` missing at the end of the statement.	Y	`,`
Protocol definition	Protocol unspecified	N	--
	`,` or `;` missing at the end of the statement.	Y	`,`
Other	Definition name other than the above	N	--

Legend:

Y: valid

N: invalid

--: Not applicable

```
Example of offline tester environment definition

# UAP definition
SPP = spp1 spp1.out spp1usr,
SPP = spp2 DUMMY    DUMMY   F,
SPP = spp3 spp3.out spp3usr D dbx,
SPP = spp4 spp4.out spp4usr,
SPP = spp5 spp5.out spp5usr N D dbx -I /betran/utf/uap/src
  spp5.out,
MHP = mhp1 mhp1.out mhp1usr,
```

```
#
# directory definition for RPC request data file
rpc_message = /betran/utf/rpcmsg,
#
# directory definition for XATMI request data file
tp_message = /betran/utf/xatmimsg,
#
# directory definition for TxRPC request data file
txrpc_message = /betran/utf/txrpcmsg,
#
# directory definition for RPC response data file
rpc_return_data = /betran/utf/rpc_return,
#
# directory definition for XATMI response data file
tp_return_data = /betran/utf/tp_return,
#
# directory definition for TxRPC response data file
txrpc_return_data = /betran/utf/tx_return,
#
# directory definition for XATMI send/receive data file
tp_converse = /betran/utf/tp_converse,
#
# directory definition for MCF receive message file
mcf_message = /betran/utf/mcfmsg,
#
# directory definition for operating command result data file
adm_call_cmd = /betran/utf/etc/call_cmd_val,
#
# directory definition for continuous execution command file
cmdfile = /betran/utf/etc,
#
# DAM file definitions
damfile = damfile1 /betran/utf/dam/damfile1,
damfile = damfile2 /betran/utf/dam/damfile2 N,
#
# TAM file definitions
tamtable = tamtable1 /betran/utf/tam/tamfile1,
tamtable = tamtable2 /betran/utf/tam/tamfile2 N,
#
# IST table definitions
isttable = isttable 128 64,
isttable = ist2 4 256,
#
# definition of function return values file
func_value_file = /betran/utf/etc/return_val,
#
# trace file definition

tracefile = /betran/utf/log/trace,
#
# protocol definition
protocol = OSI/TP;
#
```

(1) UAP definition

(a) Syntax

<pre>
{SPP|MHP}={service-group-name|service-name}
            execution-format-program-name
            user-service-definition-file-name
            [T]
            [{F|D debugger-name|[N][D debugger-name]}]{,|;}
</pre>

(b) Function

Defines the following items for the UAP to be tested by the offline tester:

- UAP type (SPP or MHP)
- Service group name
- Name of the execution format program for the service group
- Name of the user service definition file for the service group
- Whether to use the server UAP simulator
- Whether to connect a debugger

A UAP definition can only be written for an SPP or MHP. SUPs cannot be tested by the offline tester.

(c) Operands

■ `SPP|MHP`

Specify the type of service group as the definition name, as follows:

`SPP`

SPP service group

`MHP`

MHP service group

■ *service-group-name*

Specify the service group name. To use the UAP simulator with a TxRPC interface, specify the interface name that is specified in the IDL file.

■ *service-name*

Specify the service name to use the server UAP simulator with an XATMI interface.

■ *execution-format-program-name*

Specify the name of the UAP (execution format program) that executes the service group. A non-existent name can be specified when simulating service functions.

- *user-service-definition-file-name*

  Specify the name of the user service definition file that contains the environment definition for executing the service group. To use the UAP simulator with a TxRPC interface, specify the name of the user service definition file created by the `txidl` command. A non-existent name can be specified when simulating service functions.

- `T`

  Specify this operand to use the UAP simulator with a TxRPC interface. This operand can be specified only when `SPP` is specified for the service group type. If `MHP` is specified, specifying this option causes an error.

- `F`

  Specify this operand to use the server UAP simulator. If specification is omitted, the service group is activated at offline tester startup.

- `D` *debugger-name*`[`*debugger-argument*`]`

  Specify this operand to run the UAP under debugger control. Either of the following debugger names can be specified, but no name check is performed:

  `dbx`

  To use the `dbx` debugger

  `cbltd`

  To use the COBOL85/TD debugger

  The argument to be passed to the debugger can also be specified. When no argument is specified, the name specified in *execution-format-program-name* becomes the argument.

  As the argument, specify the executable file name, using the same name as specified in *execution-format-program-name*. No error occurs if the names differ, but the specified executable file name is passed to the debugger, while the offline tester uses the execution format program name to control the debugger.

- `N`

  Specify this operand to suppress activation of the service group (UAP) at offline tester startup and to activate the service group by `start` subcommand after the offline tester starts.

  When specification is omitted, the service group (UAP) is activated at offline tester startup.

(d) Note

A service group name can only be specified once. If duplicated, the first definition is valid. In the following example, `spp1` is specified twice and the definition of the second line causes an error.

Example:

```
SPP = spp1 spp1.load spp1usr,
SPP = spp1 spp2.load spp2usr,        ← Error occurs.
```

(e) Definition example

```
# UAP definition
SPP = spp1 spp1.out spp1usr,
SPP = spp2 DUMMY    DUMMY   F,
SPP = spp3 spp3.out spp3usr D dbx,
SPP = spp4 spp4.out spp4usr,
SPP = spp5 spp5.out spp5usr N D dbx -I /betran/utf/uap/src
      spp5.out,
MHP = mhp1 mhp1.out mhp1usr;
#
```

(2) Directory definition for RPC request data file

(a) Syntax

```
rpc_message = directory-name-of-RPC-request-data-file{,|;}
```

(b) Function

Defines the name of the directory that contains the RPC request data file. If the name is specified more than once, the last definition is valid.

(c) Operands

- `rpc_message`

  Write `rpc_message` as the definition name.

- *directory-name-of-RPC-request-data-file*

  Specify the pathname of the directory that contains the RPC request data file. Add the directory name if different from the directory in which the offline tester is currently executing.

(d) Definition example

```
# directory definition for RPC request data file
rpc_message = /betran/utf/rpcmsg;
#
```

(3) Directory definition for XATMI request data file

(a) Syntax

```
tp_message = directory-of-XATMI-request-data-file{,|;}
```

(b) Function

Defines the name of the directory that contains the XATMI request data file. If the name is specified more than once, the last definition is valid.

(c) Operands

- `tp_message`

  Write `tp_message` as the definition name.

- *directory-name-of-XATMI-request-data-file*

  Specify the pathname of the directory that contains the XATMI request data file. Add the directory name if different from the directory in which the offline tester is currently executing.

(d) Definition example

```
# directory definition for XATMI request data file
tp_message = /betran/utf/xatmimsg;
#
```

(4) Directory definition for TxRPC request data file

(a) Syntax

```
txrpc_message = directory-name-of-TxRPC-request-data-file{,|;}
```

(b) Function

Defines the name of the directory that contains the TxRPC request data file. If the name is specified more than once, the last definition is valid.

(c) Operands

- `txrpc_message`

  Write `txrpc_message` as the definition name.

- *directory-name-of-TxRPC-request-data-file*

  Specify the pathname of the directory that contains the TxRPC request data file. Add the directory name if different from the directory in which the offline tester is currently executing.

(d) Definition example

```
# directory definition for TxRPC request data file
txrpc_message = /betran/utf/txrpcmsg;
#
```

(5) Directory definition for RPC response data file

(a) Syntax

```
rpc_return_data = directory-name-of-RPC-response-data-file{,|;}
```

(b) Function

Defines the name of the directory that contains the RPC response data file. If the name is specified more than once, the last definition is valid.

(c) Operands

- `rpc_return_data`

  Write `rpc_return_data` as the definition name.

- *directory-name-of-RPC-response-data-file*

  Specify the pathname of the directory that contains the RPC response data file. Add the directory name if different from the directory in which the offline tester is currently executing.

(d) Definition example

```
# directory definition for RPC response data file
rpc_return_data = /betran/utf/rpc_return;
#
```

(6) Directory definition for XATMI response data file

(a) Syntax

```
tp_return_data = directory-name-of-XATMI-response-data-file{,|;}
```

(b) Function

Defines the name of the directory that contains the XATMI response data file. If the name is specified more than once, the last definition is valid.

(c) Operands

- `tp_return_data`

  Write `tp_return_data` as the definition name.

- *directory-name-of-XATMI-response-data-file*

Specify the pathname of the directory that contains the XATMI response data file. Add the directory name if different from the directory in which the offline tester is currently executing.

(d) Definition example

```
#  directory definition for XATMI response data file
tp_return_data = /betran/utf/tp_return;
#
```

(7) Directory definition for TxRPC response data file

(a) Syntax

```
txrpc_return_data = directory-name-of-TxRPC-response-data-file{,|;}
```

(b) Function

Defines the name of the directory that contains the TxRPC response data file. If the name is specified more than once, the last definition is valid.

(c) Operands

- `txrpc_return_data`

  Write `txrpc_return_data` as the definition name.

- *directory-name-of-TxRPC-response-data-file*

  Specify the pathname of the directory that contains the TxRPC response data file. Add the directory name if different from the directory in which the offline tester is currently executing.

(d) Definition example

```
#  directory definition for TxRPC response data file
txrpc_return_data = /betran/utf/tx_return;
#
```

(8) Directory definition for XATMI send/receive data file

(a) Syntax

```
tp_converse = directory-name-of-XATMI-send/receive-data-file{,|;}
```

(b) Function

Defines the name of the directory that contains the XATMI send/receive data file. If the name is specified more than once, the last definition is valid.

(c) Operands

- `tp_converse`

  Write `tp_converse` as the definition name.

- *directory-name-of-XATMI-send/receive-data-file*

  Specify the pathname of the directory that contains the XATMI send data file and XATMI receive data file. Add the directory name for the files if different from the directory in which the offline tester is currently executing.

(d) Definition example

```
# directory definition for XATMI send/receive data file
tp_converse = /betran/utf/tp_converse;
#
```

(9) Directory definition for MCF receive message file

(a) Syntax

```
mcf_message = directory-name-of-MCF-receive-message-file{,|;}
```

(b) Function

Defines the name of the directory that contains the MCF receive message file. If the name is specified more than once, the last definition is valid.

(c) Operands

- `mcf_message`

  Write `mcf_message` as the definition name.

- *directory-name-of-MCF-receive-message-file*

  Specify the pathname of the directory that contains the MCF receive message file. Add the directory name if different from the directory in which the offline tester is currently executing.

(d) Definition example

```
# directory definition for MCF receive message file
mcf_message = /betran/utf/mcfmsg;
#
```

(10) Directory definition for operating command result data file

(a) Syntax

```
adm_call_cmd = directory-name-of-operating-command-result-data-file{,|;}
```

(b) Function

Defines the name of the directory that contains the operating command result data file. If the name is specified more than once, the last definition is valid.

(c) Operands

- `adm_call_cmd`

  Write `adm_call_cmd` as the definition name.

- *directory-name-of-operating-command-result-data-file*

  Specify the pathname of the directory that contains the operating command result data file. Add the directory name if different from the directory in which the offline tester is currently executing.

(d) Definition example

```
#  directory definition for operating command result data file
adm_call_cmd = /betran/utf/etc/call_cmd_val;
#
```

(11) Directory definition for continuous execution command file

(a) Syntax

```
cmdfile = directory-name-of-continuous-execution-command-file{,|;}
```

(b) Function

Defines the name of the directory that contains the continuous execution command file. If the name is specified more than once, the last definition is valid.

(c) Operands

- `cmdfile`

  Write `cmdfile` as the definition name.

- *directory-name-of-continuous-execution-command-file*

  Specify the pathname of the directory that contains the continuous execution command file. Add the directory name if different from the directory in which the offline tester is currently executing.

(d) Definition example

```
#  directory definition for continuous execution command file
cmdfile = /betran/utf/etc;
#
```

(12) DAM file definitions

(a) Syntax

```
damfile = logical-file-name physical-file-name [N]{,|;}
```

(b) Function

Associates a logical file name with a physical file name for simulating the DAM service.

Definitions must be written for all the DAM files accessed by the UAP.

(c) Operands

- `damfile`

  Write `damfile` as the definition name.

- *logical-file-name*

  Specify the logical file name.

- *physical-file-name*

  Specify the name of the DAM file to be used by the offline tester. Add the directory name if different from the directory in which the offline tester is currently executing.

- `N`

  Specify to disable lock, regardless of any function specification.

(d) Definition example

```
# DAM file definitions
damfile = damfile1 /betran/utf/dam/damfile1,
damfile = damfile2 /betran/utf/dam/damfile2 N;
#
```

(13) TAM file definitions

(a) Syntax

```
tamtable = TAM-table-name TAM-file-name [N]{,|;}
```

(b) Function

Associates TAM table names with TAM file names for simulating the TAM service.

Definitions must be written for all the TAM files accessed by the UAP.

(c) Operands

- `tamtable`

  Write `tamtable` as the definition name.

- *TAM-table-name*

  Specify the TAM table name (name used by TAM service functions).

- *TAM-file-name*

  Specify the name of the TAM file to be used by the offline tester. Add the directory name if different from the directory in which the offline tester is currently executing.

- `N`

  Specify to disable lock, regardless of any function specifications. This operand must be specified for a UAP written in COBOL that accesses TAM files.

(d) Definition example

```
# TAM file definitions
tamtable = tamtable1 /betran/utf/tam/tamfile1,
tamtable = tamtable2 /betran/utf/tam/tamfile2 N;
#
```

(e) Note

- Each TAM file and TAM table name can only be specified once. If duplicated, the first definition is valid.

(14) Internode shared table definitions

(a) Syntax

```
isttable = IST-table-name  record-length  record-count{,|;}
```

(b) Function

Specifies an internode shared table used for the IST service simulation using a set of the internode shared table name, record length, and record count.

Define all internode shared tables accessed by the UAP. Up to 64 internode shared tables can be defined.

(c) Operands

- `isttable`

  Write `isttable` as a definition name.

- *IST-table-name*

Specify the internode shared table name. It is used for the IST service function.

- *record-length*

  Specify the record length in the internode shared table in bytes.

- *record-count*

  Specify the number of records in the internode shared table.

(d) Definition example

```
# IST table definition
isttable = isttbl1 8 12,
isttable = isttbl2 10 20;
#
```

(15) Definition of function return values file

(a) Syntax

```
func_value_file = function-return-values-file-name{,|;}
```

(b) Function

Defines the name of the file in which function return values are set. If the name is specified more than once, the last definition is valid.

(c) Operands

- `func_value_file`

  Write `func_value_file` as the definition name.

- *function-return-values-file-name*

  Specify the name of the function return values file. Add the directory name if different from the directory in which the offline tester is currently executing.

(d) Definition example

```
# definition of function return values file
func_value_file = /betran/utf/etc/return_val;
#
```

(16) Trace file definition

(a) Syntax

```
tracefile = trace-file-name{,|;}
```

(b) Function

Defines the name of the file for storing offline tester trace information. If the name is specified more than once, the last definition is valid.

(c) Operands

- `tracefile`

  Write `tracefile` as the definition name.

- *trace-file-name*

  Specify the trace file name. Add the directory name if different from the directory in which the offline tester is currently executing.

(d) Definition example

```
#  trace file definition
tracefile = /betran/utf/log/trace;
#
```

(17) Protocol definition

(a) Syntax

```
protocol = protocol-name{,|;}
```

(b) Function

Defines MCF protocol. This definition is valid only when testing a UAP written in COBOL or DML. Omit the definition for protocols other than OSI TP.

If the protocol is specified more than once, the last definition is valid.

(c) Operands

- `protocol`

  Write `protocol` as the definition name.

- *protocol-name*

  Specify the protocol name, as follows:

  `OSI/TP`

  OSI TP protocol is used.

  An error occurs if a protocol other than OSI TP is specified.

(d) Definition example

```
#  protocol definition
protocol = OSI/TP;
#
```

11.1.2 User service definition

Add the following user service definition for running the offline tester. Definition and coding are the same as for the OpenTP1 user service definition. See the manual *OpenTP1 System Definition* for details.

(1) Syntax

(a) set format

```
set service = "service-name = entry-point-name"
               [,"service-name = entry-point-name"]
               ... [set server_type = betran|xatmi]
```

(b) putenv format

```
{{[putenv environment-var-name environment-var-value]}}
```

(2) Function

Enables execution of the offline tester according to the user service definition.

(3) Operands

(a) set format

- `service = "`*service-name* `=` *entry-point-name*`"`

  For all the services in the service group, specify the service name paired with the entry point name.

  The entry point name is a function name in C or a program name or entry name in COBOL. Specify the same name as in the RPC (or XATMI) interface definition.

  See the manual *OpenTP1 Programming Guide* for details on the RPC and XATMI interface definitions and on service functions for the RPC (or XATMI) interface.

- `server_type = betran|xatmi` ~<<betran>>

  Specify whether to use OpenTP1 (RPC) or XATMI functions, as follows:

  `betran`

  Use OpenTP1 (RPC) functions.

  `xatmi`

  Use XATMI functions.

(b) putenv format

- *environment-var-name*

Set the value of the specified environment variable for the processes in the service group.

Use this format to set the COBOL environment when OpenTP1 activates a COBOL operating environment. The user can choose an environment variable for each UAP execution format program. Reference `putenv` in the standard C library.

(4) Definition example

(a) set format

```
set service     = "service = xwsvkd0100"
set server_type = betran
```

(b) putenv format

```
putenv      CBLCORE 1
```

11.1.3 Setting function return values

To enable a simulated OpenTP1 function to return a fixed value, create a function return values file and set the value in the file.

Using this file, you can also set event types for the XATMI functions `tpsend` and `tprecv` and the output data (node ID and server name) to be passed to a function used by the multi-node facility.

Definition and coding are the same as for the offline tester environment definition.

(1) Syntax

(a) Set return value

```
{function-name|program-name(request-code)} = return-value {,|;}
```

(b) Set event type

```
{tpsend|tprecv} = TPEEVENT,
{tpsend(event)|tprecv(event)} = event-type {,|;}
```

(c) Set output data

```
{function-name-for-multinode(node_id)|function-name-for-multinode(sv_name)} =
{node-ID|server-name}{,|;}
```

(2) Function

Defines a user-specified value as the return value for an OpenTP1 function. Or, defines an event type for the XATMI function `tpsend` or `tprecv`, or the output data (node ID and server name) to be passed to TP1/Multi function.

(3) Operands

(a) Set return value

- *function-name* | program-name (*request-code*)

  Specify a function name or program name (request code) for returning the value.

  function-name

  Return value for C function

  program-name (*request-code*)

  Return value for COBOL program. Set the request code in parentheses.

- *return-value* ~<1-39 alphanumerics>

  Set the return value (or return code for COBOL) to be returned by the function or program.

  Write the return value as an upper-case constant name. Use a constant name also when setting a COBOL return code for a TX function.

  Alternatively, the return value can be set as a numeric value (decimal) in the following range:

Interface	Specifiable range
C interface	-99999 to 99999
COBOL interface	0 to 99999#

Note

A specification outside the specifiable range is regarded as a character string.

#: For a TX function, specify the return code within the range -99999 to 99999.

If an undefined constant name is specified or if a numeric value is incorrectly specified (non-numeric, for example), the offline tester assumes that the function returned normally.

(b) Set event type

- `{tpsend|tprecv} = TPEEVENT`

  Indicates that the subsequent coding sets an event type.

- `tpsend(event)|tprecv(event)`

  Specify the function to which the event type applies.

`tpsend(event)`

Event type for `tpsend` function

`tprecv(event)`

Event type for `tprecv` function

- *event-type*

  Set the event type for the `tpsend` or `tprecv` function. If specification is omitted, `TPEV_SVCERR` is assumed.

(c) Set output data

- *function-name-for-multimode* (`node_id`) |*function-name-for-multimode* (`sv_name`)

  Specify the function name to which the output data applies and the output data type, as follows:

  `node_id`

  Sets the node ID as the output data.

  `sv_name`

  Sets the server name as the output data.

  The following function names and output data types can be specified:

Function name	Type of output data
dc_adm_get_nd_status_next	node_id
dc_adm_get_sv_status_next	sv_name
dc_adm_get_nodeconf_next	node_id
dc_adm_get_node_id	node_id

- *node-ID* | *server-name*

  Specify the node ID or server name.

  Node IDs and server names are associated with the sequence of multi-node functions issued by the UAP in the order in which they are specified in the function return values file.

  When a node ID or server name is omitted or incorrectly specified, that line of coding is ignored and the system processes the specifications as if the node ID or server name does not exist. Therefore, the node ID or server name is not counted.

  When the UAP issues more functions than the number of node IDs and server names in the function return values file, `DCADMER_NO_MORE_ENTRY` is returned by the excess functions (but not by the `dc_adm_get_node_id` function).

(4) Definition examples

(a) C

```
dc_jnl_ujput  = 0,
dc_dam_open   = DCDAMER_PROTO,
#dc_trn_begin = DC_OK,
dc_dam_read   = -1600,
tpsend        = TPEEVENT,
tpsend(event) = TPEV_DISCONIMM,
dc_adm_get_nd_status_next(node_id) = ND01
        :
        :
dc_logprint  = DC_OK;
```

(b) COBOL

```
CBLDCJNL(UJPUT)  = 0,
#CBLDCTRN(BEGIN) = 905,
CBLDCDAM(READ)   = 1600,
        :
        :
CBLDCLOG(PRINT)  = 1905;
```

(5) Notes

- During definition analysis, the system does not check the validity of the functions and return values or the relationships among them.
- A format error occurs when an unsupported function, a function that does not return a return value, or a function that accesses a DAM file in the offline environment is specified as a function name.
- Duplicate specifications (same function name, or a function and a program that perform the same process) are not permitted. If specifications are duplicated, the system sets the return value specified first.
- When a format error is detected during definition analysis, an error message is output and analysis continues. The table below shows whether each definition statement is valid or invalid when a format error occurs.

Format error	Valid	Assumed specification when valid
`,` or `;` missing at the end of statement.	Y	`,`
Other	N	--

Legend:

--: Not applicable

- Request codes must be those listed in the manual *OpenTP1 Programming Reference COBOL Language*. However, for the following processes, specify the request code shown below.

Description	Request code
Delete records in TAM table	`DELT`
Input TAM records	`READ`
Update or output TAM records	`WRIT`

- Specify the following return values for a COBOL program that returns a status code at normal termination:

  `CBLDCADM(STATUS)`

  Set the status code for the user server.

  `CBLDCTAM(GST)`

  Set the following values:

  `1: RO` (open status)

  `2: RC` (close status)

  `3: HL` (logical shutdown status)

  `4: HO` (error shutdown status)

  Examples:

  If `CBLDCADM(STATUS)=1` is set, the return information is:

Return value=`0`
User server status code=`1`

If `CBLDCTAM(GST)=3` is set, the return information is:

Return value=`0`
TAM table status=`HL (`*logical shutdown status*`)`

11.1.4 Setting continuous execution commands

To enable continuous execution of commands in the set sequence, create a continuous execution command file and set the commands in the file in the required execution sequence. If the `end` subcommand is included, the offline tester terminates and does not execute the remaining commands.

Commands (`read` and other subcommands) for responding to offline tester inquiries during service execution can also be set in the file. If no response subcommands are set in the file, the system waits for user input.

Definition and coding are the same as for the offline tester environment definition.

(1) Syntax

```
command-name [command-argument ...] {,|;}
```

(2) Function

Defines commands for consecutive execution by the offline tester.

(3) Operands

- *command-name*

  The following values can be specified as the command name:

  - `call`
  - `end`
  - `ps`
  - `read`
  - `start`
  - `stop`
  - `write`

  When a command other than the above is specified, a message reports that a command error has occurred. The command is ignored and processing continues.

- *command-argument*

  Set the command arguments for the specified command.

(4) Definition example

```
call ser1 sppsub1 a_data,
call ser2 mcfsub b_data+c_data,
call ser3 sppsub2 d_data,
read rtn_data,
#call ser1 sppsub1 b_data,
         :
         :

end;
```

(5) Notes

- When a format error is detected during definition analysis, an error is output and analysis continues. The table below shows whether each definition statement is valid or invalid when a format error occurs.

Format error	Valid	Assumed specification when valid
`,` or `;` missing at the end of statement.	Y	`,`
Other	N	--

Legend:

--: Not applicable

- Each command is checked at execution when the `cmdauto` subcommand is actually entered.

11.1.5 Creating stubs

Stubs are required for UAPs (SPPs and MHPs) that provide services in an RPC, XATMI, or TxRPC environment.

Stubs for UAPs with the RPC or XATMI interface are created by a stub generator from the *RPC (or XATMI) interface definition file* which contains the RPC (or XATMI) interface definitions. For UAPs with the TxRPC interface, stubs or server UAP templates are created using an OpenTP1 command with the *Interface Definition Language file*. Translate the stubs using a C compiler, then link the stubs to the server UAP's object file.

Create stubs for the offline tester in the same way as for a job UAP. See the manual *OpenTP1 Programming Reference C Language* for details.

11.2 User-created files

Table 11-2 lists the files that the user must create to use the offline tester.

Table 11-2: List of user-created files

File type		Use and contents	Time of creation	Deleted by	Time of deletion
Service request data files	RPC request data file	Stores request data passed to the server UAP when using the client UAP simulator with an RPC interface.	Before service request	User	Any
	XATMI request data file	Stores request data passed to the server UAP when using the client UAP simulator with an XATMI interface.	Before service request	User	Any
	TxRPC request data file	Stores request data passed to the server UAP when using the client UAP simulator with a TxRPC interface.	Before service request	User	Any
Service response data files	RPC response data file	Stores data returned as the service result when using the server UAP simulator with an RPC interface.	At activation of the simulate SPP	User	Any
	XATMI response data file	Stores data returned as the service result when using the server UAP simulator with an XATMI interface.	At activation of the simulate SPP	User	Any
	TxRPC response data file	Stores data returned as the service result when using the server UAP simulator with a TxRPC interface.	At activation of the simulate SPP	User	Any
XATMI receive data file		Stores data received by the `tprecv` function	Before service request	User	Any
MCF receive message file		Stores messages passed to the MHP when using the MCF simulator.	Before service request	User	Any
Operating command result data file		Stores data returned to the UAP as the execution result when using the operating command simulator.	Before service request	User	Any
User file		Used when the DAM or TAM facility is used.	Before offline tester startup	User	Any

Note

All user-created files for the online tester can be used without modification, except the following:

TMI receive data file

MCF receive message file

Operating command result data file

11.2.1 Service request data files

(1) RPC request data file

An RPC request data file stores the data passed to the service function for a service requested when using the client UAP simulator with an RPC interface. A single file contains one set of data.

(a) File structure

Data length	Response area length	Data

(b) File contents

Item	Position	Length (bytes)	Contents
Data length	0	4	Length of the data to be passed to the service function. (0 - specified value of `DCRPC_MAX_MESSAGE_SIZE`)
Response area length	4	4	Length of the response area to be passed to the service function. (1 - specified value of `DCRPC_MAX_MESSAGE_SIZE`)
Data	8	*n*	Data to be passed to the service function.

(c) Notes

- The items in the RPC request data file are related to the service function arguments as follows:

$$\text{Service function }(\underbrace{\texttt{in}}_{1.},\underbrace{\texttt{in\_len}}_{2.},\texttt{out},\underbrace{\texttt{out\_len}}_{3.})$$

1. Data
2. Data length
3. Response area length

- An RPC request data file for the online tester can also be used.
- Do not use a plus sign (+) in the file name. Also, do not use `ps` or `end` as the file

name.

- An error occurs when the specified data is less than the specified data length. Data that exceeds the data length is truncated.

(2) XATMI request data file

An XATMI request data file stores the data passed to the service function for a requested service when using the client UAP simulator with an XATMI interface. A single file contains one set of data.

(a) File structure

Call type	Buffer type	Buffer subtype	Flags	Data length	Data

(b) File contents

Item	Position	Length (bytes)	Contents

Call type	0	8	Type of function calling a service:
			`call`
			call from `tpcall` function
			`acall`
			call from `tpacall` function
			`connect`
			call from `tpconnect` function

Buffer type	8	8	Buffer type, specified as one of the following character strings: • `X_OCTET` • `X_COMMON` • `X_C_TYPE`
Buffer subtype	16	16	Buffer subtype, specified as a string of up to 16 characters. Specify a null character when specifying `X_OCTET` as the buffer type.

Item	Position	Length (bytes)	Contents
Flags	32	4	Flags to be passed to the service function, specified as a hexadecimal. 0x00000000L 0 0x00000004L TPNOREPLY 0x00000008L TPNOTRAN 0x00000100L TPNOCHANGE 0x00000800L TPSENDONLY 0x00001000L TPRECVONLY
Data length	36	4	Length of the data to be passed to the service function (0-524288). Specify zero when no data is passed. The buffer type and subtype specifications are ignored when zero is specified.
Data	40	*n*	Data to be passed to the service function

(c) Notes

- The items in the XATMI request data file are related to the service function arguments as follows:

```
void tpservice(svcinf)
     TPSVCINFO *svcinf;

     struct TPSVCINFO {
               char name[32];  ......1.
               char *data;     ......2.
               long len;       ......3.
               long flags;     ......4.
               int cd;         ......5.
            }
```

1. Service name
2. Address at which the data mapped to `buff_type` and `sub_type` is stored
3. Length of the data shown by `data`
4. Flags (specified flags stored in bit strings)
5. Interactive descriptor (stores zero)

- An XATMI request data file for the online tester can also be used.

- Do not use a plus sign (+) in the file name. Also, do not use `ps` or `end` as the file name.
- An error occurs when the specified data is less than the specified data length. Data that exceeds the data length is truncated.
- The response data area is reallocated according to the buffer type and buffer subtype in the response data.
- When the buffer type and subtype are specified, the values specified for the data length and data must be the same as the data structure value defined for the stubs.

  Boundary alignment is performed for the data structure specified for the stubs (the total length is an integer multiple of 4). For this reason, the user must consider the alignment portion when creating an XATMI request data file.

(3) TxRPC request data file

A TxRPC request data file stores the data passed to the service function for a requested service when using the client UAP simulator with a TxRPC interface. A single file contains one set of data.

(a) File structure

Major version	Minor version	Data length	Data

(b) File contents

Item	Position	Length (bytes)	Contents
Major version	0	2	Major version number specified in the interface definition of the `txidl` command. Specify zero to omit this specification.
Minor version	2	2	Minor version number specified in the interface definition of the `txidl` command. Specify zero to omit this specification.
Data length	4	4	Length of the data to be specified for a data part (0 to specified value of `DCRPC_MAX_MESSAGE_SIZE` - 16).
Data	8	*n*	Argument data to be passed to the service function. When setting an address in the argument, set the contents of the area indicated by the address. Set character string `#NULL##` if the address is null.

(c) Notes

- The following shows data contents of the TxRPC request data file and how the service function arguments are related to the data received by the arguments.

Data contents

Data of argument 1 (n bytes)	Data of argument 2 (m bytes)	Data of argument 3 (k bytes)	...

```
        ↓
Service function = Data received by the arguments (data length)
arguments
Argument 1       = Data of argument 1 (n bytes)
Argument 2       = Data of argument 2 (m bytes)
Argument 3       = Data of argument 3 (k bytes)
    :                      :
```

Example:
Data contents of the TxRPC request data file to be passed to service function `serviceA`

00000004	007b	234e554c232300

Service function arguments and data received by the arguments

```
serviceA(p1, p2, p3)
long   p1;  (Data received by p1 = 4)
short  p2;  (Data received by p2 = 123)
char  *p3;  (Data received by p3 = NULL)
```

- Do not use a plus sign (+) in the file name. Also, do not use `ps` or `end` as the file name.
- An error occurs when the specified data is less than the specified data length. Data that exceeds the data length is truncated.
- UAP operations are not guaranteed when the data contains an error.

11.2.2 Service response data files

(1) RPC response data file

When using the server UAP simulator with an RPC interface, the RPC response data file stores the response data returned to the client UAP when a service request is made to the simulate SPP. A single file contains one set of data.

(a) File structure

Data length	Data

(b) File contents

Item	Position	Length (bytes)	Contents
Data length	0	4	Length of the data to be returned to the UAP making the service request. (0-2147483647)
Data	4	*n*	Data to be returned to the UAP making the service request.

(c) Notes

- The items in the RPC response data file are related to the arguments of the service request function (`dc_rpc_call` function) as follows:

```
dc_rpc_call(.....,in,in_len,out,out_len)
                            1.
```

1. Data

- An RPC response data file for the online tester can also be used.
- Do not use a plus sign (+) in the file name. Also, do not use `ps` or `end` as the file name.
- An error occurs when the specified data is less than the specified data length. Data that exceeds the data length is truncated.

(2) XATMI response data file

When using the server UAP simulator with an XATMI interface, the XATMI response data file stores the response data returned to the client UAP when a service request is made to the simulate SPP. A single file contains one set of data.

(a) File structure

Buffer type	Buffer subtype	Service termination code	Return code	Data length	Data

(b) File contents

Item	Position	Length (bytes)	Contents
Buffer type	0	8	Buffer type, specified as one of the following character strings: • `X_OCTET` • `X_COMMON` • `X_C_TYPE`
Buffer subtype	8	16	Buffer subtype, specified as a string of up to 16 characters. Specify a null character when specifying `X_OCTET` as the buffer type.

Item	Position	Length (bytes)	Contents

Service termination code	24	4	One of the following hexadecimal values of `rval` in the `tpreturn` function. The value is set in the `tperrno` area.
			0x04000000L
			`TPSUCCESS`
			0x20000000L
			`TPFAIL`

Return code	28	4	Hexadecimal value of `rcode` in the `tpreturn` function. The value is set in the `tpurcode` area.
Data length	32	4	Length of the data to be returned to the UAP making a service request. (0-524288) Specify zero when no data is passed. The buffer type and subtype specifications are ignored when zero is specified.
Data	36	*n*	Data to be returned to the UAP making the service request.

(c) Notes

- The items in the XATMI response data file are related to the arguments of the service termination function (`tpreturn` function) as follows:

```
tpreturn(rval,rcode,data,len,.....)
         1.   2.    3.   4.
```

1. Service termination code
2. Return code
3. Data stored in the buffer allocated by buffer type and subtype
4. Data length

- An XATMI response data file for the online tester can also be used.
- Do not use a plus sign (+) in the file name. Also, do not use `ps` or `end` as the file name.
- An error occurs when the specified data is less than the specified data length. Data that exceeds the data length is truncated.
- When the buffer type and subtype are specified, the values specified for the data length and data must be the same as the data structure value defined for the stubs.

  Boundary alignment is performed for the data structure specified for the stubs (the total length is an integer multiple of 4). For this reason, the user must consider the alignment portion when creating an XATMI response data file.

(3) TxRPC response data file

When using the server UAP simulator with a TxRPC interface, the TxRPC response data file stores the response data returned to the client UAP when a service request is made to the simulate SPP. A single file contains one set of data.

(a) File structure

System area	Data length	Return value	Data

(b) File contents

Item	Position	Length (bytes)	Contents
System area	0	12	Area used by the offline tester. Do not use this area.
Data length	12	4	Total length of the data to be specified for a data part and of the return value (0 to specified value of `DCRPC_MAX_MESSAGE_SIZE` - 16).
Return value	16	*m*	Return value of the service function. The data type and size are specified in the interface definition of the `txidl` command. Do not specify a return value for the `void` type service function.
Data	16+*m*	*n*	Argument data to be returned to the client. Specify an argument for which the `out` attribute is specified in the parameter declaration of the interface definition of the `txidl` command. When setting an address in the argument, set the contents of the area indicated by the address. Set character string `#NULL##` if the address is null.

(c) Notes

- The following shows data contents of the TxRPC response data file and how the service function arguments are related to the data received by the arguments.

Data contents

Data of argument 1 [`out` attribute] (*n* bytes)	Data of argument 3 [`out` attribute] (*m* bytes)	...

↓

Service function arguments		Data received by the arguments (data length)
Argument 1 (`out` attribute)	=	Data of argument 1 (*n* bytes)
Argument 2 (`in` attribute	=	No data received
Argument 3 (`out` attribute)	=	Data of argument 3 (*m* bytes)
:		:

Example:
Data contents of the TxRPC request data file to be passed to service function `serviceA`

00000004	007b	234e554c232300

Service function arguments and data received by the arguments

```
serviceA(p1,p2,p3,p4)
long  p1;   [out attribute]   (Data received by p1 = 4)
short p2;   [out attribute]   (Data received by p2 = 123)
short p3;   [in attribute]    (Data received by p3 = None)
char *p4;   [out attribute]   (Data received by p4 = NULL)
```

- Do not use a plus sign (+) in the file name. Also, do not use `ps` or `end` as the file name.
- An error occurs when the specified data is less than the specified data length. Data that exceeds the data length is truncated.
- UAP operations are not guaranteed when the data contains an error.

11.2.3 XATMI receive data file

An XATMI receive data file stores the messages received by the `tprecv` function in the UAP. A single file can contain a number of data items which are passed consecutively to the `tprecv` function.

(1) File structure

Common area	Buffer type	Buffer subtype	Event flag	Data length	Data
Common area	Buffer type	Buffer subtype	Event flag	Data length	Data
:	:	:	:	:	:
Common area	Buffer type	Buffer subtype	Event flag	Data length	Data

(2) File contents

Item	Position	Length (bytes)	Contents
Common area	0	36	Area shared with the XATMI send data file. Specify a space or null character.

Item	Position	Length (bytes)	Contents
Buffer type	36	8	Buffer type, specified as one of the following character strings: • X_OCTET • X_COMMON • X_C_TYPE
Buffer subtype	44	16	Buffer subtype, specified as a string of up to 16 characters. Specify a null character when specifying X_OCTET as the buffer type.
Event flag	60	4	One of the following hexadecimal values as the string to be passed to the tprecv function: 0x00000000L 0 0x00000001L TPEV_DISCONIMM 0x00000002L TPEV_SVCERR 0x00000004L TPEV_SVCFAIL 0x00000008L TPEV_SVCSUCC 0x00000020L TPEV_SENDONLY
Data length	64	4	Length of the data to be passed to the tprecv function (0-524288). Specify zero when no data is passed. The buffer type and subtype specifications are ignored when zero is specified.
Data	68	*n*	Data to be passed to the tprecv function

(3) Notes

- The items in the XATMI receive data file are related to the arguments of the message receive function (tprecv function) as follows:

```
tprecv(.....,data,len,.....,revent)
             1.   2.        3.
```

1. Data stored in the buffer allocated by buffer type and subtype
2. Data length
3. Event flag

- XATMI receive data files for the online tester cannot be used.
- Do not use a plus sign (+) in the file name. Also, do not use ps or end as the file

name.

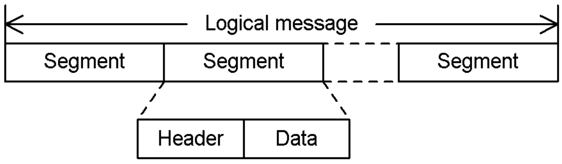
- An error occurs when the specified data is less than the specified data length. Data that exceeds the data length is truncated.
- Create the receive data in execution units. If the `tprecv` function is executed more than once in a service, create all the data required for the number of executions. If the `tprecv` function is executed more times than the number of data items, the system assumes that data from the `tpreturn` function was received and an error occurs at each execution that exceeds the number of data items.

  The XATMI receive data file opens and closes by service unit.
- When the buffer type and subtype are specified, the values specified for the data length and data must be the same as the data structure value defined for the stubs.

  Boundary alignment is performed for the data structure specified for the stubs (the total length is an integer multiple of 4). For this reason, the user must consider the alignment portion when creating an XATMI receive data file.

11.2.4 MCF receive message files

A logical message can contain one or more segments. A segment consists of a header part containing the segment information and a data part which is the message text.



There are five types of segments:

- Single segment

  Segment in a logical message consisting of one segment only
- First segment

  First segment in a logical message consisting of multiple segments
- Middle segment

  One of the middle segments in a logical message consisting of multiple segments
- Last segment

  Last segment in a logical message consisting of multiple segments
- Header segment

  Segment prefixed to two concatenated messages

Specify the segment type in the header part.

An MCF receive message file stores the messages received by the UAP in an MCF function (`dc_mcf_receive`, `dc_mcf_recvsync`, or `dc_mcf_sendrecv`). Create one logical message per file. Two messages can be concatenated if a header segment is used.

(1) File structure

■ Logical message consisting of one segment only

<table>
<tr><td colspan="2">Single segment</td></tr>
<tr><td>Header</td><td>Data</td></tr>
</table>

■ Logical message consisting of multiple segments

<table>
<tr><td colspan="2">First segment</td><td colspan="2">Middle segment</td><td colspan="2">Middle segment</td><td></td><td colspan="2">Last segment</td></tr>
<tr><td>Header</td><td>Data</td><td>Header</td><td>Data</td><td>Header</td><td>Data</td><td></td><td>Header</td><td>Data</td></tr>
</table>

■ Header segment

<table>
<tr><td colspan="2">Header segment</td></tr>
<tr><td>Header</td><td>Data</td></tr>
</table>

(2) File contents

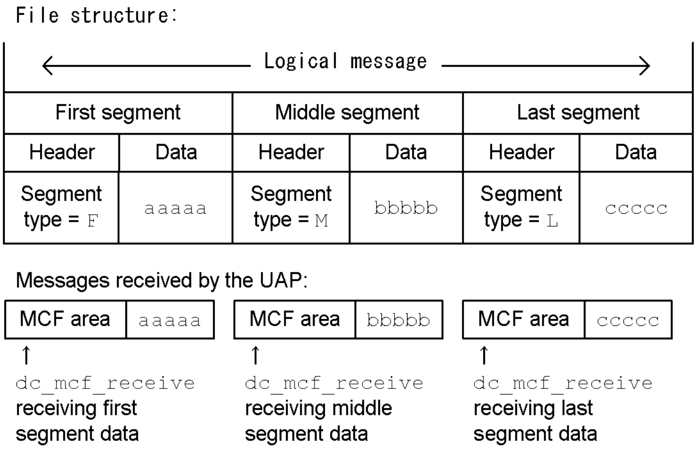
Item		Position	Length (bytes)	Contents
Header	Input/output logical terminal name	0	9	Logical terminal name (including final null character) to be passed in MCF functions. Specify the same name for each segment of a multiple-segment message.
	Map name	9	9	Map name (including final null character). Specify the same name for each segment of a multiple-segment message. This specification is valid only for functions that return a map name.
	Reserved	18	9	Null character

	Segment type	27	1	One of the following characters:
				`F`
				First segment
				`M`
				Middle segment
				`L`
				Last segment
				`O`
				Single segment
				`H`
				Header segment

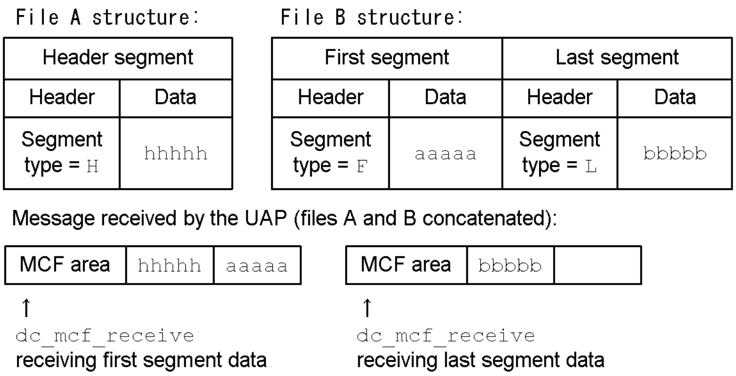
	Message length	28	4	Message length (0-2147483647)
Data	Message	32	*n*	The data in the segment, of the specified message length

(3) Notes

- The following shows how the items in an MCF receive message file are related to message receive requests from a UAP via an MCF function.



- By concatenating header segments, data created in another file can be combined with the first or single segment and passed together to the UAP. The following shows how a header segment is related to a message receive request from a UAP by an MCF function.



- The following shows the relationships between the segment type specified in the segment header for a service request to an MHP and the file type at execution.

■ Logical message consisting of one segment only

When segment type `F`, `M`, or `L` is specified, the message is handled in the same way as when `O` is specified and no error occurs.

Segment type	File type
F	Handled as an MCF receive message file.
M	
L	
O	
H, #	Handled as an invalid file specification. The system makes a file name inquiry.

Legend:

#: Specification other than F, M, L, O, or H.

■ Logical message consisting of multiple segments

When segment type L, H, or O is specified, the MHP regards the message as completed and ignores any subsequent segments. Segment type F is handled in the same way as segment type M.

Segment type			Segments received by MHP
First segment	Middle segment	Last segment	
F	M	L	F, M, L
F	L	M	F, L#1
F	O	L	F, L#2
M	M	L	M, M, L
L	M	F	L#3
O	O	O	O#3
F	L	M	F, L#1
F	M	H	No segments received.#4
X	M	L	
F	X	L	
F	M	X	
H	F	L	H#3
H	O	O	H#3

Legend:

X: Specification other than F, M, or L.

#1: M is ignored.

#2: L is ignored.

#3: The middle and subsequent segments are ignored.

#4: Handled as an invalid file specification. The system makes a file name inquiry.

■ Files concatenated by header segment

Files can only be concatenated when H is specified as the segment type. Otherwise, the file specifications are ignored.

Segment type (combinations for concatenation)	File type
H + (file beginning with F)	Handled as a concatenated MCF receive message file.
H + (file beginning with M)	
H + (file beginning with L)	
H + (file beginning with O)	
H + (file beginning with #)	
F, M, L, or O + (file beginning with any segment type)	The file following + is ignored.
# + (file beginning with any segment type)	Handled as an invalid file specification. The system makes a file name inquiry.

Legend:

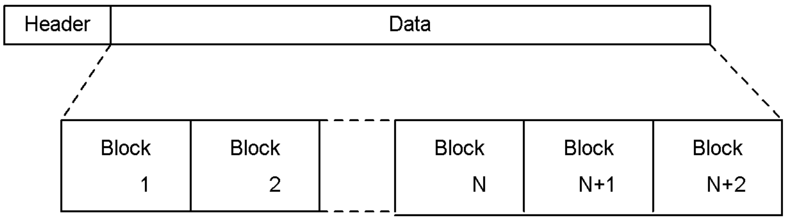
#: Specification other than F, M, L, O, or H.

Do not use a plus sign (+), space, or tab code in the file name. Also, do not use `ps` or `end` as the file name.

11.2.5 DAM file

A DAM file stores DAM file data for the offline tester when the DAM service simulator is used. DAM files are created by using an editor or by creating and executing a program that uses the `dc_dam_create` function provided by the offline tester.

(1) File structure



(2) File contents

Item		Position	Length (bytes)	Contents
Header	File name	0	64	DAM file name. The specification is not checked.
	File name	64	4	Length of one block (0-32760)
	Total no. of blocks	68	4	Total number of blocks in the data part (1-2147483647)
	Unused	72	2	Null character

	Shutdown status	74	2	Specify one of the following:
				`0x0000`
				Not shutdown (normal)
				`0x0001`
				Logical shutdown
				`0x0002`
				Error shutdown

	Reserved	76	20	Null character
Data	Block	96	*n*	Any data, specified by block.

(3) Note

The system does not check whether the total number of blocks in the data part is the same as the actual block count. An error occurs at data access if the actual block count is less.

11.2.6 TAM file

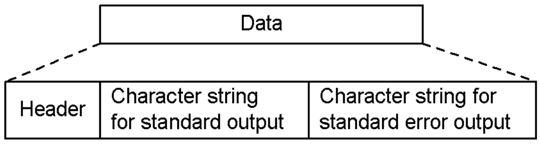
A TAM file stores TAM file data for the offline tester when using the TAM service simulator. TAM files are created from a TAM data file by entering the offline tester's `utftamcre` command (see Section *13.1 Operating commands for running tests*).

Create a TAM data file in the same way as a TAM file used by TP1/FS/Table Access. Or, use a job TAM file as is. See the manual *OpenTP1 Operation* for creating job TAM files.

11.2.7 Operating command result data file

An operating command result data file stores the data returned to the UAP as the command execution result when using the operating command simulator. A single file contains one data item.

(1) File structure



(2) File contents

Item		Position	Length (bytes)	Contents
Header	Operating command result code	0	4	Result code value set in the `stat` argument of the `dc_adm_call_command` function
	Character string length for standard output	4	4	Length of character strings (including null characters) output to standard output (0-2147483647)
	Character string length for standard error output	8	4	Length of character strings (including null characters) output to standard error output (0-2147483647)
Character string for standard output		12	*n*	Value set in the `outmsg` argument of the `dc_adm_call_command` function. (Includes the final null character. If no null characters are added, the last character is replaced with a null character.) The specified value is ignored when zero is specified as the character string length for standard output.
Character string for standard error output		--	*n*	Value set in the `errmsg` argument of the `dc_adm_call_command` function. (Includes the final null character. If no null characters are added, the last character is replaced with a null character.) The specified value is ignored when zero is specified as the character string length for standard error output.

Legend:

--: Not applicable

(3) Notes

- An operating command result data file for the online tester cannot be used.
- Add a null character at the end of a standard output string and a standard error output string. If no null character is added for such strings, the last character in the string is replaced by a null character. If you specify 0 as the string length, the character string is ignored even if it is specified.
- Do not use a plus sign (+) in the file name. Also, do not use `ps` or `end` as the file name.
- When issuing operating commands by SEND statement in a DML, specify the data part as follows:

  Character string length for standard output:

    Specify 0.

  Character string length for standard error output:

    Specify 0 (when standard error output is not available).

11.3 Creating files

This section explains how to create test data definition files for simplifying later creation of tester files, and provides a list of the files generated by the offline tester.

11.3.1 Test data definition file

By creating a *test data definition file*, the user can easily create tester files using the tester file creation facility.

A test data definition file can have any name. The following tester files can be created from a test data definition file:

- RPC request data file
- XATMI request data file
- TxRPC request data file
- RPC response data file
- XATMI response data file
- TxRPC response data file
- XATMI receive data file
- MCF receive message file
- Operating command result data file

(1) Syntax

```
#comment          ]1.
start tester-file-ID  tester-file-kind  output-destination-file-name  ]2.
keyword  =  input-data          ]5.
keyword  =  input-data
sep        ]3.
keyword  =  input-data
    :             :
    :             :
keyword  =  input-data
end        ]4.
```

Note that the italicized numbers in the box above correspond to the numbers under *(3) Description* below.

(2) Function

Allows the tester file creation command to create a tester file after the definition of test data needed for the tester file.

One line in the definition file can contain up to 512 bytes including a carriage return code.

(3) Description

1. Comment statement

   Write a comment statement.

   - *comment*

     Write a comment in a line.

2. `start` statement

   Declare the beginning of input data for a tester file. This statement is required for declaring input data in each tester file.

   When input data is created for multiple tester files in a test data definition file, the `end` statement shows the end of input data of one tester file.

   - *tester-file-ID* ~<up to 14 alphanumerics>

     Specify an ID for identifying input data in each tester file described in the test data definition file. The ID must be unique in a test data definition file.

   - *tester-file-kind*

     Specify a tester file kind. Available tester file kinds are:

     `RRQ`

     RPC request data file

     `XRQ`

     XATMI request data file

     `TRQ`

     TxRPC request data file

     `RRT`

     RPC response data file

     `XRT`

     XATMI response data file

     `TRT`

     TxRPC response data file

     `XRV`

     XATMI receive data file

`NRV`

MCF receive message file

`COM`

Operation command result data file

- *output-destination-file-name* ~<pathname>

  Specify the name of a tester file made of input data.

  When a test data definition file specifies input data of multiple tester file kinds, specify different output destination file names for the file kinds.

  If the same output destination file name is used for input data with different tester file kinds, test data is appended to the specified file. Though this is not an error, the created tester file may be unavailable for testing. If the existing file name is specified, test data is appended to that file.

3. `sep` statement

   Specify a data separator when creating a tester file that contains multiple data entries.

   If a file contains multiple data entries for the offline tester, however, only the first data entry takes effect, ignoring the second or later data.

   The `sep` statement is specifiable for creating the following tester files.

   - XATMI receive data file
   - Operation command result data file

4. `end` statement

   Declare the end of input data in a tester file. This statement is required for every input data in each tester file.

5. Input data definition statement

   Define input data in each tester file.

   Input data includes fixed information data and user data. The *fixed information data* provides predetermined information to be specified. The *user data* (with the keyword `data`) can contain anything the user specifies. In a set of test data, specify all fixed data prior to user data.

   Input data cannot duplicate in a set of test data. In the operation command result data file, however, specify user data twice for setting standard output character string data and standard error output character string data.

   - *keyword*

     Specify a keyword for identifying data specific to each tester file. Space characters or tab codes are ignored if specified before or after the keyword.

- *input-data*

  Specify input data for the keyword. Space characters or tab codes are ignored if specified before or after the keyword.

  For details about the input data formats for specifying fixed information data, see the tables in *(5) Formats for the input data corresponding to the keywords of tester files*, below

(4) Required settings for specifying user data as input data

The following describes an input data format for specifying user data.

(a) Setting the user data length

Specify the length of the entire user data as fixed information data in the following format.

`data_len=`*bytes*

If the data specified as user data is larger than the data length, the system truncates the data and issues a message. If the data is smaller than the data length, nothing is appended to it.

Example:

```
data_len=5
data='1234567'
```

→ Data:

31	32	33	34	35

```
data_len=5
data='123'
```

→ Data:

31	32	33	00	00

(b) Initializing user data

Using the tester file creation command, initialize the user data for the specified user data length.

(c) Setting character data

Set character data in the following format:

`data='`*data*`'`

Do not add a null character to the end of character data.

Example:

`data='12345'` → Data:

31	32	33	34	35

(d) Setting binary data

Set binary data in the following format:

`data=`*data*

Data can be written in decimal and hexadecimal notation, as follows:

- Decimal notation

  Specify the value as is.
- Hexadecimal notation

  Prefix `0x` to the value.

Example:

`data=5` → Data: Decimal 5

`data=0x05` → Data: Hexadecimal 5

Data is set with the `int` type.

(e) Setting hexadecimal code format data

Set hexadecimal code data in the following format:

`data=(code)0x`*data*

In *data*, write n bytes of 2n-digit data using hexadecimal code. The user can write as many number of bytes as required within the maximum length of a line.

Write a value of `0x00-0xff` for one byte of data.

The data is assumed as binary data written in hexadecimal notation if `(code)` is not specified.

Example:

`data=(code)0x1234` → Data: | 12 | 34 |

(f) Setting special characters

The system processes a carriage return code, tab code, null character, single quotation mark (`'`), and backslash (\) to be special characters in character data. Enter these characters as follows.

Character	Notation
Carriage return	\n
Tab code	\t
Null character	\0

Character	Notation
'	\'
\	\\

(g) Setting data to be read from the file

Use the following format when using data as user data read from the file.

`data=(file)` *file-pathname*

Example:

`data=(file)/tmp/datafile` ➜ Use data in `/tmp/datafile`.

(h) Setting the beginning of data

Specify the beginning of data as follows.

`data=[`*offset-from-start-of-user-data*`]` *data*

Example:

```
data_len=10
data=[2]'1234'
```

→ Data:

00	00	31	32	33	34

(i) Setting a format for multiple data types

`data=`*data*
`=`*data*
:
:

Example:

`data=0x00000001` ➜ First data

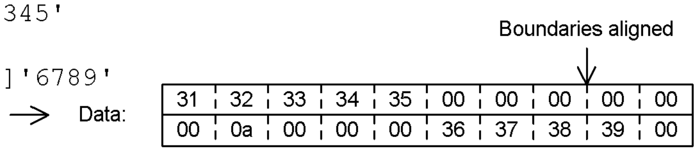
`='ABCDEF'` ➜ Second data

(j) Adjusting the boundary

When multiple data types are described, adjacent data types may differ from each other. This time the tester file creation command sets data by automatically adjusting the boundary for the preceding data. However, no boundary adjustment takes place when:

- User data is read from the file.
- The beginning of user data is set.
- Hexadecimal code format data is set.

```
data_len=20
data='12345'
    =10
    =[15]'6789'
```



(5) Formats for the input data corresponding to the keywords of tester files

The following tables list the keywords and the formats of the corresponding input data for each tester file. For the type of information to be specified, see the description of each tester file in Section *11.2 User-Created files*.

Table 11-3: RPC request data file keywords and input data formats

Keyword	Specified information	Description
`out_len`	Response area length	Before `data`, specify the response area length in decimal or hexadecimal placed in the `dc_rpc_call` function.
`data_len`	Data length	Before `data`, specify the user data length in decimal or hexadecimal passed to the server UAP with the `dc_rpc_call` function.
`data`	Data	Specify the user data passed to the server UAP with the `dc_rpc_call` function.

Table 11-4: XATMI request data file keywords and corresponding input data formats

Keyword	Specified information	Description
`call_kind`	Call kind	Before `data`, specify one of the following character strings as a function type for service request. • `call` • `acall` • `connect`
`buff_type`	Type	Before `data`, specify one of the following character strings as a buffer type. • `X_OCTET` • `X_COMMON` • `X_C_TYPE`
`sub_type`	Subtype	Before `data`, specify a subtype within 16 characters. Example: `sub_type=subtype01`

Keyword	Specified information	Description
flag	Flag	Before data, specify the following character string as a flag to be passed to the service function. Separate multiple flags with a vertical line (|). • 0 • TPNOREPLY • TPNOTRAN • TONOCHANGE • TPSENDONLY • TPRECVONLY
data_len	Data length	Before data, specify the user data length in decimal or hexadecimal to be passed to the server UAP with the tpcall, tpacall, or tpconnect function.
data	Data	Specify user data to be passed to the server UAP with the tpcall, tpacall, or tpconnect function.

Table 11-5: TxRPC request data file keywords and corresponding input data format

Keyword	Specified information	Description
version	Version number	Before data, specify the version number in decimal or hexadecimal specified in the interface definition of the txidl command. This information is optional. If omitted, zero is assumed. The range of specification is 0-65535. Example: version = : The version is 0.0. version = 2: The version is 2.0. version = 3.2: The version is 3.2.
data_len	Data length	Before data, specify the user data length in decimal or hexadecimal to be passed to the server UAP.
data	Data	Specify user data to be passed to the server UAP.

Table 11-6: RPC response data file keywords and corresponding input data formats

Keyword	Specified information	Description
data_len	Data length	Before data, specify the user data length in decimal or hexadecimal to be passed to the client UAP on service termination.
data	Data	Specify user data returned to the client UAP on service termination.

Table 11-7: XATMI response data file keywords and corresponding input data formats

Keyword	Specified information	Description
`buff_type`	Type	Before `data`, specify one of the following character strings as a buffer type. • `X_OCTET` • `X_COMMON` • `X_C_TYPE`
`sub_type`	Subtype	Before `data`, specify a subtype within 16 characters. Example: `sub_type=subtype01`
`rval`	Service termination code	Before `data`, specify one of the following character strings as a service termination code. • `TPSUCCESS` • `TPFAIL`
`rcode`	Return code	Before `data`, specify the return code in decimal or hexadecimal.
`data_len`	Data length	Before `data`, specify the user data length in decimal or hexadecimal passed to the client UAP on service termination.
`data`	Data	Specify user data returned to the client UAP on service termination.

Table 11-8: TxRPC response data file keywords and corresponding input data format

Keyword	Specified information	Description
`data_len`	Data length	Before `data`, specify the user data length in decimal or hexadecimal to be passed to the client UAP.
`svc_rtn`	Return value	Before `data`, specify the return value in decimal or hexadecimal to be passed to the client UAP.
`data`	Data	Specify user data to be passed to the client UAP.

Table 11-9: XATMI receive data file keywords and input data formats

Keyword	Specified information	Description
`buff_type`	Type	Before `data`, specify one of the following character strings as a buffer type. • `X_OCTET` • `X_COMMON` • `X_C_TYPE`

Keyword	Specified information	Description
sub_type	Subtype	Before data, specify a subtype within 16 characters. Example: sub_type=subtype01
event	Event flag	Before data, specify one of the following character strings as an event flag passed to the tprecv function. • 0 • TPEV_DISCONIMM • TPEV_SVCERR • TPEV_SVCFAIL • TPEV_SVCSUCC • TPEV_SENDONLY
data_len	Data length	Before data, specify the user data length in decimal or hexadecimal passed to the tprecv function.
data	Data	Specify user data passed to the tprecv function.
sep	sep statement	When specifying data for multiple services, place a sep statement at the end of data for one service. Do not place this statement after the last data.

Note

When specifying data for multiple services, repeat buff_type and succeeding data.

Table 11-10: MCF receive message file keywords and corresponding input data formats

Keyword	Specified information	Description
termname	I/O logical terminal name	Before data, specify an I/O logical terminal name within 8 characters passed to the dc_mcf_receive function.
mapname	Map name	Before data, specify a map name within 8 characters passed to the dc_mcf_receive function.

Keyword	Specified information	Description
seg_kind	Segment type	Before data, specify one of the following characters as a segment type passed to the dc_mcf_receive function. • F • M • L • O • H Specify these characters in any of the following orders when there is data for multiple segments. • F...M...L • F...F...L • M...M...L • L • H • O
data_len	Message length	Before data, specify the user data length of the segment in decimal or hexadecimal passed to the dc_mcf_receive function.
data	Message	Specify user data of the segment passed to the dc_mcf_receive function.

Note

When specifying data for multiple segments, repeat seg_kind and succeeding data.

Table 11-11: Operation command result data file keywords and corresponding input data formats

Keyword	Specified information	Description
status_code	Operation command result code	Before data, specify a result code in decimal returned by the operation command.
outsize	Standard output character string length	Before data, specify the message length in decimal or hexadecimal the operation command outputs to standard output.
errsize	Standard error output character string length	Before data, specify the message length in decimal or hexadecimal the operation command outputs to standard output error.

Keyword	Specified information	Description
data	Standard output character string	Specify a message with character data the operation command outputs to standard output.
data	Standard error output character string	Specify a message with character data the operation command outputs to standard output error.
sep	sep statement	When specifying data for multiple commands, place a sep statement at the end of data for one command. Do not place this statement after the last data.

11.3.2 Files created by the offline tester

Table 11-12 lists the files created by the offline tester.

Table 11-12: List of files created by offline tester

File type	Use and contents	Time of creation	Deleted by	Time of deletion
XATMI send data file	Stores data sent by the tpsend function.	At execution of the tpsend function	User	Any
Temporary memory data file	Stores data updated by the dc_mcf_tempput function and acquired by the dc_mcf_tempget function in the UAP when using the MCF simulator.	In the dc_mcf_tempput and dc_mcf_tempget functions#1	Offline tester#2	At execution of the dc_mcf_contend function
Trace file	Collects offline tester trace information.	When the offline tester (UAP) collects the first trace information.	User	Any

#1: Created in the /tmp directory, with the logical terminal name acquired by the dc_mcf_receive function as the file name. Not created when the same file name already exists in the /tmp directory.

#2: When not running a UAP that issues the dc_mcf_contend function, the user can delete the file at any time.

Chapter
12. Test Execution

This chapter explains how to run a test with the offline tester.

This chapter contains the following sections:

12.1 Creating UAPs
12.2 Starting and ending an offline test
12.3 Activating and terminating UAPs
12.4 Service requests
12.5 Creating tester files
12.6 Continuous command execution
12.7 Debugger connection
12.8 Editing offline tester trace information
12.9 Notes on running tests

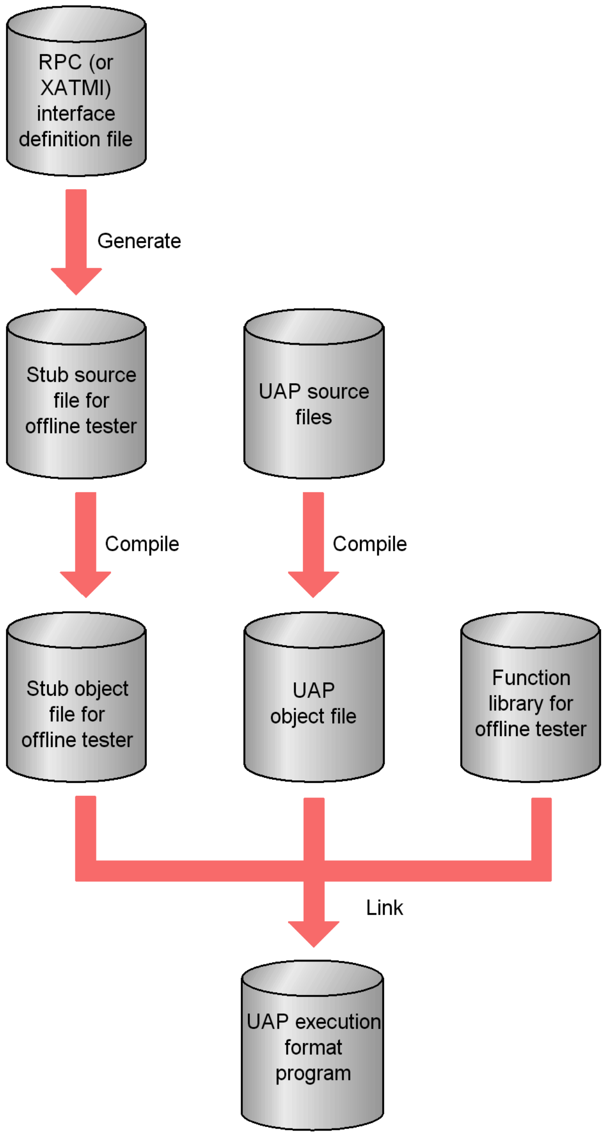
12.1 Creating UAPs

12.1.1 Creating UAP execution format programs

(1) Creating UAP execution format program with the RPC or XATMI interface

Figure 12-1 shows the procedure for creating a UAP execution format program with the RPC or XATMI interface.

Figure 12-1: Procedure for creating UAP execution format program with the RPC or XATMI interface



To create the stub source program for creating a UAP execution format program with an RPC or XATMI interface, use the `stbmake` command with an RPC (or XATMI) interface definition file. See the manual *OpenTP1 Programming Guide* for details on the `stbmake` command.

The following examples show how to generate stubs.

Example:

Generate stubs from an RPC interface definition file.

```
stbmake spp1stb.def
              1.
```

1. RPC interface definition file

   (The name of the source file generated in this example is `spp1stb_sstb.c`.)

Example:

Generate stubs from an XATMI interface definition file.

```
stbmake -x spp1stb.def
                 1.
```

1. XATMI interface definition file

   (The name of the source file generated in this example is `spp1stb_stbx.c` and the header file name is `spp1stb_stbx.h`.)

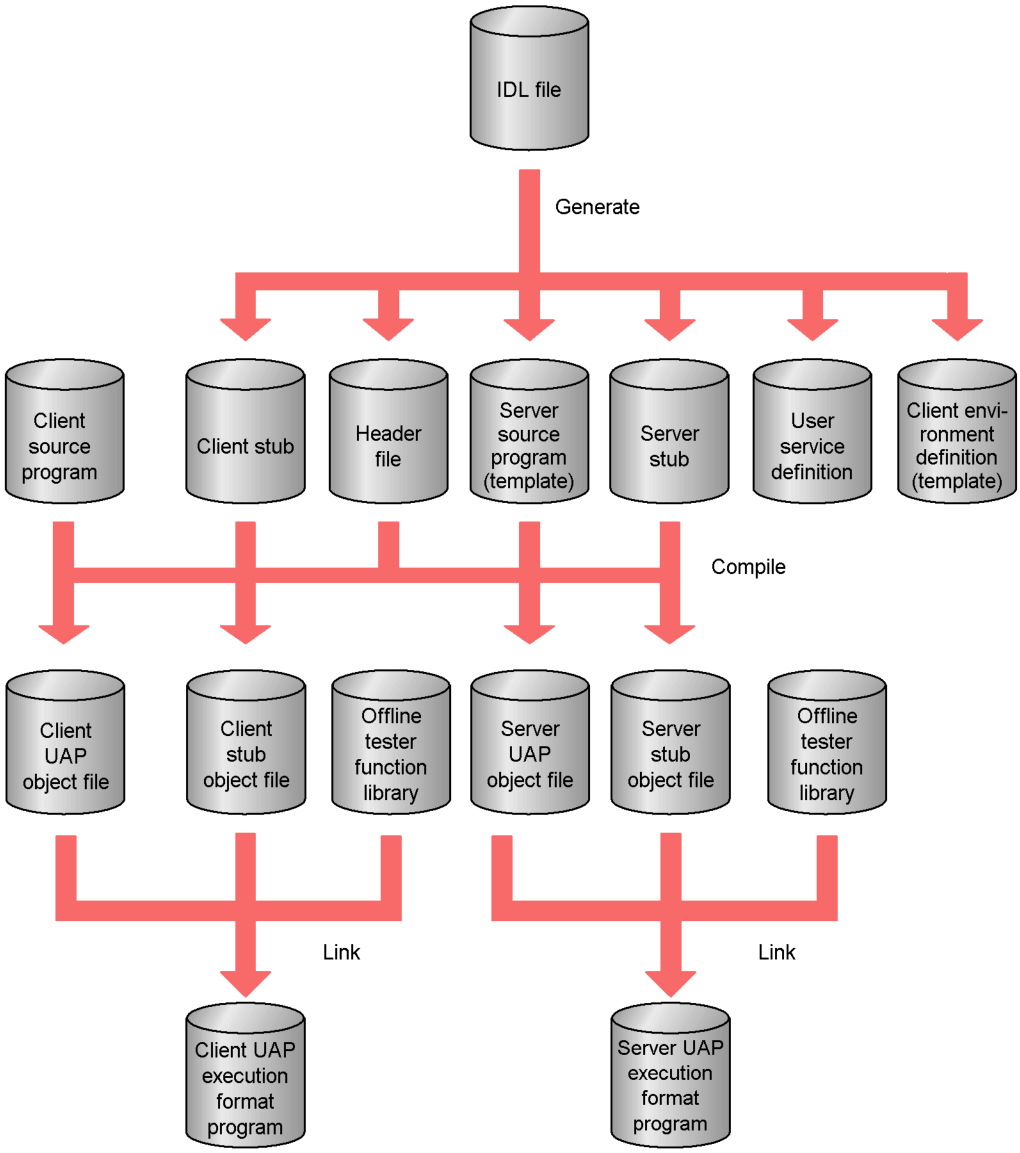
After generating the stubs, compile the stubs and UAP (C or COBOL). Use the header file provided by TP1/Server Base.

After compilation, link the stub object file and UAP object file to the simulation functions library provided by the offline tester.

(2) Creating UAP execution format program with a TxRPC interface

Figure 12-2 shows the procedure for creating a UAP execution format program with a TxRPC interface.

Figure 12-2: Procedure for creating UAP execution format program with the TxRPC interface



To create the client stub or server stub source program or server UAP templates for creating a UAP execution format program with the TxRPC interface, use the OpenTP1 `txidl` command with the *Interface Definition Language (IDL) file*. See the manual

OpenTP1 Programming Guide for details on the `txidl` command.

The following example shows how to create stubs and a template.

Example:

Generate stubs from an IDL file

```
txidl spp1.idl
           1.
```

1. Interface Definition Language file name

   The following six files are generated in this example:

   `spp1_cstub.c` (Client stub source)

   `spp1_sstub.c` (Server stub source)

   `Cspp1` (User service definition for client)

   `Sspp1` (User service definition for server)

   `spp1.h` (Header file)

   `spp1.c` (Server source program template)

After generating the files, code the UAP based on the template and then compile the stubs and UAP (C). Use the header file provided by OpenTP1. See the manual *OpenTP1 Programming Reference C Language* for how to create the UAP.

After compilation, link the stub object file and UAP object file to the simulation functions library provided by the offline tester. For a client UAP, link the client stub object file. For a server UAP, link the server stub object file.

12.2 Starting and ending an offline test

To start the offline tester, execute the `utfstart` command. In the command, specify the name and option parameters of the offline tester environment definition file that defines the execution conditions.

Starting the offline tester activates the service groups specified in the offline tester environment definition file. A prompt (`?>`) for command input is displayed as each UAP executes its `main` function and issues a function for starting services (`dc_rpc_mainloop` or `dc_mcf_mainloop` function). Execute an offline tester subcommand in response to the prompt.

At offline tester startup, a number of service groups are activated at the same time. That is, a number of UAPs may run in parallel.

To end the offline tester, execute the `end` subcommand when the prompt is displayed.

12.3 Activating and terminating UAPs

When the offline tester is used, the offline tester controls activation and termination of UAPs (service groups) instead of OpenTP1. At offline tester startup, all the UAPs are activated except those for which activation at tester startup is suppressed by a specification in the offline tester environment definition.

When the offline tester has completed startup, the `start` subcommand can be executed to activate a UAP that has not yet activated or a UAP that terminated due to an error.

Terminating the offline tester terminates all the active UAPs. To terminate one UAP, execute the `stop` subcommand.

12.4 Service requests

A service can be requested in either of the following ways:

- By issuing a service request (`dc_rpc_call` function) in the program
- By executing the `call` subcommand

Execute the `call` subcommand after the UAP (service group) has activated.

12.5 Creating tester files

To create a tester file, execute the `utffilcre` command.

The procedure for creating tester files from a test data definition file is the same as for the online tester (see Subsection *11.3.1 Test data definition file* in Part IV).

12.6 Continuous command execution

To execute offline tester commands continuously, execute the `cmdauto` subcommand. Specify the name of the continuous execution command file as the command argument.

Subcommands for user responses can also be set in the file. If a command in the file contains an error, the command is ignored or the offline tester prompts for command input.

At completion of a UAP process (debugger process) other than execution of the `stop` subcommand, the offline tester asks the user whether to continue or cancel continuous command execution. The offline tester also waits for user response if no subcommand is specified at any point during continuous command execution.

12.7 Debugger connection

To run UAPs under debugger control, specify debugger connection in the offline tester environment definition. Parameters required for the debugger (the directory for the test UAP source file) must also be set in the definition.

Debugger connection is executed by the `main` function of the UAP. After control is passed to the debugger and initialization is completed, enter a program start command to start the program. When the program completes execution, terminate the debugger. The debugger cannot be restarted.

Two types of debuggers can be used:

- `dbx`
- `cbltd` (COBOL85/TD)

Follow the procedure for using each debugger.

12.8 Editing offline tester trace information

Offline tester trace information is collected in a trace file according to the output specifications (output file, content to be output, and so on) set as options at offline tester startup. Collected trace information can be output for each service or service group by executing the `utftrcpic` command.

The `dc_rpc_open` function executes the processing, such as opening the trace file, to prepare for trace collection. Therefore, trace information for functions issued before the `dc_rpc_open` function cannot be collected. Also, trace information cannot be collected for the following simulation functions for DAM file access:

- `dc_dam_create`
- `dc_dam_get`
- `dc_dam_iclose`
- `dc_dam_iopen`
- `dc_dam_put`

For a UAP written in COBOL, API trace information may not be output if the request code, DML, or other specification is incorrect. In such cases, the system outputs error message KFCA20016-E or KFCA20018-E. If the DML is incorrect, error information is also output by the COBOL compiler and the program may terminate abnormally.

When UAPs run in parallel during `main` function execution, for example, each output line may contain mixed trace information. To avoid this problem, activate each service group at a different point.

12.9 Notes on running tests

This section describes points to remember when running tests with the offline tester.

12.9.1 Notes on the offline tester

(1) Processing after abnormal termination of the offline tester

The offline tester uses pipe and shared memory facilities to control processes.

If the offline tester is terminated abnormally in an irregular manner by pressing the interrupt key, for example, the shared memory area and any temporary files in current use are saved as allocated. The offline starter can still be restarted, but the shared memory area and temporary files should be deleted if resource efficiency is likely to be affected.

The offline tester uses the following names for temporary files:

- `shm`*xxxx* (in the `/tmp` directory)
- `cpi`*xxxx* (in the `/tmp` directory)
- `ppi`*xxxx* (in the `/tmp` directory)
- *tttttttt* (in the `/tmp` directory)
- *aaaaaaaaxxxx* (in the `/tmp` directory)

Legend:

xxxx

Hexadecimal display of process ID at execution

tttttttt

Same name as logical terminal name returned when the `dc_mcf_receive` function receives the first segment. (Up to 8 characters)

aaaaaaaa

Same name (up to 8 characters) as the IST table name specified in the offline tester environment definition file.

Example:

- `shm4e7`
- `cpi3e9`
- `ppi3e8`
- `termnalA`

If the offline tester terminates abnormally, the UAP process and debugger process (if a debugger is connected) may still be active, depending on the termination timing. In such cases, execute the kill command to terminate the processes.

(2) Upper limits of the offline tester

Table 12-1 sets out the upper limits of the offline tester.

Table 12-1: Upper limits of offline tester

Item	Description	Upper limit	Processing when upper limit is exceeded
UAP startup wait time	Time from generation to activation of a UAP process (`dc_rpc_open` function) when starting the offline tester or executing the `start` subcommand	60 minutes	An error message is output and the process is forcibly terminated.#1
UAP stop wait time	Time from a termination request to actual termination of a UAP process when stopping the offline tester or executing the `stop` subcommand. Or, if debugger connection is specified for the UAP, time until the debugger process terminates.	10 minutes	Forcibly terminate the UAP process or debugger process.
Command line length	Length of command lines in offline tester subcommands. Or, length of definition lines in the continuous execution command file	254 bytes	An error message is output and the command is rejected.
Definition line length	Length of definition lines in the offline tester environment definition file or in the function return values file	510 bytes	An error message is output. The line is ignored and definition analysis continues.#2
Length of pathname information	Length of directory names and pathnames specified in the offline tester environment definitions and commands	255 bytes	An error message is output and the specification is ignored.
Number of function return value definitions	Number of function return values defined in the function return values file	200	An error message is output. Subsequent lines are ignored and processing continues.
Number of DAM files	Number of DAM files opened by the `dc_dam_open` or `dc_dam_create` function in a UAP	200	An error message is output and the `dc_dam_open` or `dc_dam_create` function returns an error value.

Item	Description	Upper limit	Processing when upper limit is exceeded
Number of TAM files	Number of TAM files opened by the `dc_tam_open` function in a UAP	200	An error message is output and the `dc_tam_open` function returns an error value.
Number of `dc_rpc_call` functions	Number of UAP executions of the `dc_rpc_call` function with `DCRPC_NOWAIT` specified when `dc_rpc_poll_any_replies` is not issued	200	An error message is output and the `dc_rpc_call` function returns an error value.
Number of synchronous message send/receive functions	Number of executions of the `dc_mcf_sendrecv` and `dc_mcf_recvsync` function in a service	100	An error message is output and the `dc_mcf_sendrecv` or `dc_mcf_recvsync` function returns an error value.

#1: Excluding UAPs for which debugger connection is specified.

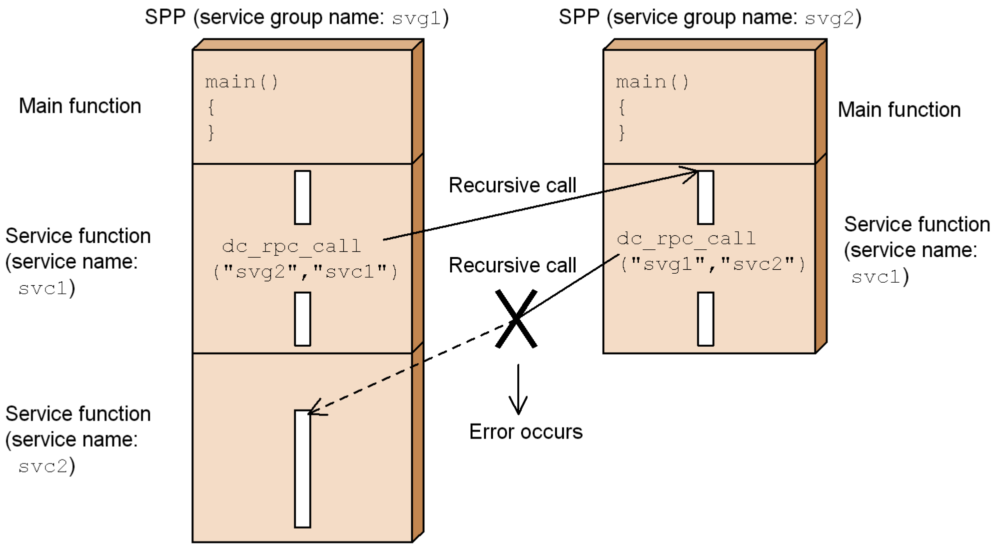
#2: When definition analysis is completed, the system waits for command input to continue or cancel offline tester startup.

(3) Recursive calls between service groups

Using the offline tester, the `dc_rpc_call` function can be used to execute nested services within a service function. However, a service can only be called once within nested services that belong to the same service group.

Figure 12-3 illustrates the use of recursive calls using the offline tester.

Figure 12-3: Recursive calls using the offline tester



(4) Functions that cannot be used before or after service calls

The offline tester outputs an error message and an error value is returned when one of the following functions is issued before or after a service call (before the `dc_rpc_mainloop` or `dc_mcf_mainloop` function is issued or after the `dc_rpc_mainloop` or `dc_mcf_mainloop` function returns):

- `dc_rpc_call`
- `dc_adm_call_command`
- MCF function other than `dc_mcf_open`, `dc_mcf_close`, or `dc_mcf_mainloop`

(5) User exit routine functions

Of the functions related to user exit routines, the offline tester does not support the `dc_mcf_svstart` function. To test a UAP that includes this function, create and link a dummy function of the same name.

(6) Accessing TAM tables in DAM file access functions

The offline tester does not support accessing of TAM tables in DAM file access functions. Operation is not guaranteed if access is attempted.

(7) Transaction processing

The offline tester does not support processing that depends on whether the process is inside or outside a transaction.

(8) Event notification by tpsend function

The `tpsend` function cannot be used for event notification in interactive service requests using an XATMI interface. To check UAP events, use the function return values file.

(9) IST table access

The IST simulation facility of the offline tester stores IST table contents in a temporary file for reference or update. This may cause a file access error that does not occur otherwise.

When this error occurs, the system issues an error message. The function that caused the file access error returns with an error condition. The return value corresponds to one of error return values returned by that function.

12.9.2 Notes on files

(1) Lock of DAM files and TAM files

Locks can be placed on each DAM or TAM file. This means that a deadlock may occur between UAPs which can normally be executed in parallel without a deadlock occurring (because the UAPs have exclusive access to separate blocks within a DAM file, for example).

If a deadlock occurs, take one of the following actions:

- Suppress lock in the offline tester environment definition.
- Suppress update by specifying the `-c` option in the `utfstart` command.
- Prevent the UAPs from running in parallel by entering the `start` subcommand to start the UAPs sequentially after the offline tester starts.

(2) Number of batch processing blocks in DAM files

The offline tester processes files by block, regardless of the value set as the number of batch processing blocks when issuing the `dc_dam_create` or `dc_dam_iopen` function. However, no processing is performed when the specified value is less than zero.

(3) Closing DAM files and TAM files

Always issue the `dc_dam_close` or `dc_tam_close` function after issuing the `dc_dam_open` or `dc_tam_open` function.

If the service group is terminated without issuing the `dc_dam_close` or `dc_tam_close` function, a duplicate open error or lock error may occur at the DAM (or TAM) file when the service is re-executed. If an error occurs, enter the `stop` subcommand to terminate the service (or service group), then enter the `start` subcommand to reactivate the service.

(4) Lock of TAM files used by COBOL UAPs

COBOL UAPs cannot place locks on TAM files. When creating a UAP in COBOL, specify suppression of lock in the TAM definition statement in the offline tester environment definition.

If suppression is not specified, a lock error may occur when a service that accesses a TAM file is restarted. If an error occurs, enter the `stop` subcommand to terminate the service (or service group), then enter the `start` subcommand to restart the service.

12.9.3 Notes on UAPs

(1) Infinite looping of a UAP

As the offline tester does not perform timer monitoring, offline tester responses may cease if the UAP goes into a infinite loop and makes no further responses. In this case, execute the `kill` command from another window to forcibly terminate the UAP process.

Operation is not guaranteed if the `kill` command is used to forcibly terminate a process other than a UAP that has stopped issuing responses.

Chapter

13. Operating Commands

This chapter explains how to use the operating commands and subcommands of the offline tester.

This chapter contains the following sections:

13.1 Operating commands for running tests
13.2 Subcommands for running tests

13.1 Operating commands for running tests

Table 13-1 lists the operating commands for running offline tests.

Table 13-1: List of operating commands for offline testing

Command name	Function
`utfdamcre`	Creation of offline tester DAM file
`utffilcre`	Tester file creation
`utfstart`	Offline tester startup
`utftamcre`	Creation of offline tester TAM files
`utftrcpic`	Retrieval of offline tester trace information from a file

13.1.1 utfdamcre (creation of offline tester DAM file)

(1) Syntax

`utfdamcre` *block-length block-count DAM-file-name* [*input-file-name*]

(2) Function

Reads a DAM data file and creates an offline tester DAM file.

(3) Command arguments

- *block-length* ~((sector length x *n* - 8))

  Specify the block length of a DAM file.

- *block-count* ~((1-2147483647))

  Specify the number of blocks in a DAM file to be created. The DAM file size will be (block length x block count + 96) bytes.

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

  Specify the name of a DAM file to be created.

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

  Specify the name of a file that stores data to be output to the DAM file. Omitting this specification outputs null data to the DAM file.

(4) Notes

- When an error occurs during `utfdamcre` command execution, the DAM file remains allocated. Before reexecuting the `utfdamcre` command, use the `rm` command to delete the DAM file.

- The following operations take place when the block count specified for the `utfdamcre` command differs from the block count in the input file.

Specified block count > block count in the input file

The system outputs blocks of null data to the end of the DAM file.

Specified block count < block count in the input file

The system stops reading blocks from the input file, issues message KFCA20789-W, then terminates the `utfdamcre` command.

13.1.2 utffilcre (tester file creation)

(1) Syntax

`utffilcre -e` *test-data-definition-file-name*

(2) Function

Creates tester files from the specified test data definition file.

(3) Option

- `-e` *test-data-definition-file-name* ~<pathname>

  Specify the name of the test data definition file that contains the input data for the tester files.

13.1.3 utfstart (offline tester startup)

(1) Syntax

`utfstart [-s] [-l] [-i] [-f] [-g] [-d] [-c]` *offline-tester-environment-definition-file-name*

(2) Function

Starts the offline tester according to the definitions in the offline tester environment definition file.

(3) Options

- `-s`

  Outputs service function names and return information to standard output as offline tester trace information.

  This option is ignored when the `-i` option is specified.

- `-l`

Outputs function argument information, as well as service function names and return information, to standard output as offline tester trace information.

This option is ignored when the `-i` option is specified.

- `-i`

  Suppresses output of offline tester trace information.

- `-f`

  Outputs offline tester trace information to standard output and to a trace file.

  When an existing trace file is specified, the information is added at the end of the existing data. If the specified trace file does not exist, the offline tester creates the file.

  This option is ignored when the `-g` option is specified.

- `-g`

  Outputs offline tester trace information to standard output and to a trace file.

  When an existing trace file is specified, the file is recreated and information is written from the head of the file. If the specified trace file does not exist, the offline tester creates the file.

- `-d`

  Outputs all the contents to standard output when the function argument information consists of a data area (buffer, for example).

  When this option is omitted, 20 bytes of information are output.

  This option is valid only when the `-l` option is specified.

- `-c`

  Suppresses update of DAM files and TAM files when using the DAM service or TAM service.

  When this option is omitted, DAM files and TAM files are updated.

(4) Command argument

- *offline-tester-environment-definition-file-name* ~<pathname>

  Specify the name of the offline tester environment definition file containing the test environment.

(5) Note

When all the options are omitted, the `-l` option is assumed.

13.1.4 utftamcre (creation of offline tester TAM files)

(1) Syntax

```
utftamcre  -r record-length -l key-area-length -k key-start-position
           -m max-record-count [-t] [-u hash-entry-usage] [-s]
           [-d TAM-data-file-name] TAM-file-name
```

(2) Function

Inputs the TAM data file and creates a TAM file for the offline tester.

(3) Options

- `-r` *record-length* ~((1-2147483647))

  Specify the record length of the TAM file.

- `-l` *key-area-length* ~((1-2147483647))

  Specify the key length.

- `-k` *key-start-position*

  Specify the offset to the key position from the head of the record.

  An error occurs if a non-zero value is specified in this option and the `-s` option is also specified. The record length of the management part of the TAM file is: (*record-length*) - (*key-area-length*).

- `-m` *max-record-count* ~((1-2147483647))

  Specify the maximum number of records in TAM tables.

- `-t`

  Creates TAM tables in tree structure.

  TAM tables are created in hash structure when this option is omitted, provided the `-u` option is specified.

- `-u` *hash-entry-usage* ~((1-100))

  Specify the usage percentage of indexes to be used as hash areas.

  An error occurs if this option is specified with the `-t` option.

- `-s`

  Specify this option to delete the key area from record contents.

- `-d` *TAM-data-file-name* ~((255))

  Specify the name of the TAM data file in up to 255 characters. An error occurs if the number of characters is over 255 or if the specified name is the same as the

TAM file name. Check the two name specifications.

(4) Command argument

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

  Specify the name of the TAM file to be created by the command.

(5) Notes

- An error occurs when the data length of the TAM data file exceeds (*record-length*) x (*maxd-record-count*).
- When the data length of the TAM data file cannot be evenly divided by the record length specified by the `-r` option, the excess data is truncated and is not stored in the TAM file.

13.1.5 utftrcpic (retrieval of offline tester trace information)

(1) Syntax

```
utftrcpic   trace-file-name  service-group-name
            [service-name  [data-file-name]]
```

(2) Function

Retrieves offline tester trace information by key from a trace file and outputs the information to standard output.

(3) Command arguments

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

  Specify the name of the trace file that contains the offline tester trace information.

- *service-group-name* ~<identifier of 1-31 characters>

  As the key information, specify the name of the service group that contains the trace information to be retrieved.

- *service-name* ~<identifier of 1-31 characters>

  As the key information, specify the name of the service that contains the trace information to be retrieved.

  When specification is omitted, trace information is retrieved by service group.

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

  Specify the name of a data file as the key information if you want to restrict the retrieved trace information to a specific data file used at service execution.

(4) Output format

```
18:41:01 Function=dc_dam_read [CBLDCDAM(READ)]                                   1.
         file descriptor(IN)=00000008
         DAM key(IN)=first-block-number   last-block-number                      2.
                     00000000             00000000
         number of DAM keys(IN)=00000001
         input data(OUT)=
           00000000 00000000 00000000 00000000 00000000                          3.
           00000000 00000000 00000000 00000000 00000000
         input buffer length(IN)=000001f8
         option flags(IN)=00000009
                        DCDAM_REFERENCE [request:R]                              4.
                        DCDAM_NOEXCLUSIVE [exclusive:N]
         return value=DC_OK(000000) [00000]                                      5.
```

1. Time and function information:
    - Time at which the service group was activated (hour:minute:second)
    - Name of C function
    - Name of COBOL program
    - Request code
    - DML statement name
2. Argument information:

   `(IN)` indicates contents specified with the function argument by the UAP. `(OUT)` indicates contents returned by the function to the UAP. `arg name (OUT)=NULL` is displayed when the address of the character string area is a null character.
3. Information on data and data length:

   Data contents are displayed for the specified data length in 40 bytes per line. The format when a specification is incorrect or incomplete is as follows:

   Example:

   When the data address is a null character:

```
data name(IN)=NULL
```

   When the data length is zero:

```
data name(IN)=
```

4. Option flag information:
    - Option flag name

- COBOL flag name
- COBOL flag type

If a specification is incorrect, the code of the incorrect flag is displayed and `***` is displayed as the COBOL flag name and flag type.

Example:

```
option flags(IN)=00000001
              DCDAM_FILE_EXCLUSIVE [exclusive:B]
              00000006 [***]
```

5. Return value information:
   - Definition name of C return value
   - Decimal display of C return value
   - Decimal display of COBOL return code

Output example

```
KFCA20001-I   Process was created. Service group name=svg2(xdb)

15:18:56 function=dc_rpc_open(svg2)  [CBLDCRPC(OPEN)]
         option flags(IN)=00000000
                         DCNOFLAGS                                      1.

KFCA20000-I   Offline tester was activated. Tue May 31 15:18:56 1994
              return value=DC_OK(000000)  [00000]

15:18:56 function=dc_rpc_mainloop(svg2)  [CBLDCRSV(MAINLOOP)]
         option flags(IN)=00000000
                         DCNOFLAGS

?>call svg2 svc5 xd03km                                                 2.

15:19:18 service start(svc5)
          buffer type(IN)=X_OCTET
          buffer length(IN)=00000000
          data(IN)=NULL
          option flags(IN)=00000000
                          DCNOFLAGS

15:19:18 function=tpalloc
         buffer type(IN)=X_OCTET
         buffer subtype(IN)=NULL
         buffer length(IN)=0000008c
         return value=DC_OK(ADDRESS)

15:19:18 function=tprealloc
         buffer length(IN)=00000096
         return value=DC_OK(ADDRESS)

15:19:18 function=tpcall
         service name(IN)=svc1
         send buffer type(IN)=X_OCTET
         send buffer length(IN)=0000008c
         data (IN)=
           00000000 00000000 00000000 00000000 00000000
           00000000 00000000 00000000 00000000 00000000
           00000000 00000000 00000000 00000000 00000000
           00000000 00000000 00000000 00000000 00000000
           00000000 00000000 00000000 00000000 00000000
           00000000 00000000 00000000 00000000 00000000
           00000000 00000000 00000000 00000000 00000000      3.
           00000000 00000000 0000
         receive buffer type(IN)=X_OCTET
         receive buffer length(IN)=0000008c
         option flags(IN)=00000008
                         TPNOTRAN
```

```
read(svg2:svc5:crm_rtn)?>read xd04km                                      ]4.
            receive buffer type(OUT)=X_OCTET
            receive buffer length(OUT)=00000000
            data(OUT)=
            return value from service(OUT)=999999
            return value=DC_OK(000000)

15:19:27  function=tpreturn
            end code(IN)=TPSUCCESS
            return value(IN)=000000
            buffer length(IN)=00000000
            data(IN)=NULL
            option flags(IN)=00000000
                         DCNOFLAGS

15:19:27  service end(svc5)

?>end                                                                        ] 5.

15:19:31  return value(svg2)=DC_OK(000000)  [00000]

15:19:31  function=dc_rpc_close(svg2)  [CBLDCRPC(CLOSE)]                       6.
            option flags(IN)=00000000
                         DCNOFLAGS
```

1. Message indicating offline tester startup and trace information collected at SPP startup
2. Trace information collected at subcommand input (test start)
3. Trace information collected at service execution
4. Trace information collected at data file read
5. Trace information collected at subcommand input (test end)
6. Trace information collected at SPP termination when offline tester ends

(5) Notes

- Offline tester trace information is retrieved from the start to the end of each service.
- The retrieval range of the trace information differs according to the user response when prompted for input of the `read` or `write` subcommand or for input of a file name during service execution.

  The table below shows how the user response determines the retrieval range.

Input prompt	Response (command input)	Trace information retrieval
`read` or `write` subcommand	`read` or `write` subcommand	Trace information is also retrieved after subcommand input.
	`ps` subcommand	Command input information and the command execution result are not retrieved.
	`end` subcommand	Trace information is not retrieved after subcommand input.
	Invalid command	Command input information and error messages are not retrieved.
File name	`ps` subcommand	Command input information and the command execution result are not retrieved.
	`end` subcommand	Trace information is not retrieved after subcommand input.
	Command other than `ps` or `end` subcommand	Trace information is also retrieved after subcommand input.

13.2 Subcommands for running tests

Table 13-2 lists the subcommands for running offline tests.

Table 13-2: List of subcommands for offline testing

Command name	Function
`call`	Service request
`cmdauto`	Continuous command execution
`end`	Offline tester termination
`ps`	Test status display
`read`	Input of tester file name to offline tester
`start`	Service group activation
`stop`	Service group termination
`write`	Input of tester file name to offline tester

13.2.1 call (service request)

(1) Syntax

```
call  service-group-name  service-name
{ RPC-request-data-file-name|
  XATMI-request-data-file-name|
  TxRPC-request-data-file-name|
  MCF-receive-message-file-name
                [+ MCF-receive-message-file-name]}
```

(2) Function

Activates the SPP or MHP process corresponding to the specified service group name and executes the service function for the specified service name.

(3) Command arguments

- *service-group-name* ~<identifier of 1-31 characters>

  Specify the name of the service group that contains the service to be activated.

- *service-name* ~<identifier of 1-31 characters>

  Specify the name of the service to be activated.

- *RPC-request-data-file-name* ~<pathname>

Specify the name of the RPC request data file that contains the input data received by the first service function when requesting the service from an RPC interface SPP.

- *XATMI-request-data-file-name* ~<pathname>

  Specify the name of the XATMI request data file that contains the input data received by the first service function when requesting the service from an XATMI interface SPP.

- *TxRPC-request-data-file-name* ~<pathname>

  Specify the name of the TxRPC request data file that contains the input data received by the first service function when requesting the service from a TxRPC interface SPP.

- *MCF-receive-message-file-name* ~<pathname>

  Specify the name of the MCF receive message file that contains the data for input to the UAP by the MCF function when requesting the service from an MHP.

  To create concatenated messages, specify a second MCF receive message file, prefixed with a plus sign (+).

(4) Notes

- The service group name must be defined in the offline tester environment definition and the service name must be defined in the user service definition.
- If a tester file cannot be accessed, or if the file contents are incorrect, the next prompt is displayed for file name input. When concatenation of MCF receive message files is specified, if an error occurs at one of the files, specify both of the file names in the specification.

  Format

```
file(group1:service1)?>
     1.      2.
```

1. Service group name
2. Service name

13.2.2 cmdauto (continuous command execution)

(1) Syntax

```
cmdauto continuous-execution-command-file-name
```

(2) Function

Executes offline tester commands in sequence, according to the contents of the continuous execution command file.

(3) Command argument

- *continuous-execution-command-file-name* ~<pathname>

  Specify the name of the continuous execution command file containing the commands to be executed successively.

13.2.3 end (offline tester termination)

(1) Syntax

```
end
```

(2) Function

Terminates active service groups and ends the offline tester.

(3) Note

This command sets normal return for the `dc_rpc_mainloop` function of each service group. If the UAP process (or debugger process when using debugger connection) does not complete within 10 minutes, the command forcibly terminates the UAP process (or debugger process). However, if the command is entered while the system is waiting for input of the `read` subcommand or file name, the service group terminates normally only after the offline tester issues the `dc_rpc_close` function.

13.2.4 ps (test status display)

(1) Syntax

```
ps
```

(2) Function

Displays the status of processes running under the offline tester.

(3) Output format

```
18:23:43  PID   Type  Service-group-name   S  D  DPID
          1925  SPP   group1               R  *  *****
          ****  SPP   group1               E  *  *****
          1927  SPP   group2               R  D  0013
          1928  SPP   group3               F  *  *****
          1929  MHP   group4               F  *  *****

   1.        2.    3.      4.              5. 6.   7.
```

1. Time at which the `ps` subcommand was executed (hour:minute:second)

2. UAP process ID.

   `*****` is displayed when the process is inactive.

3. Service group type code:

   `SPP`

   Indicates an SPP.

   `MHP`

   Indicates an MHP.

4. Service group name

5. Process status:

   `R`

   Indicates that the service group process is active.

   `E`

   Indicates that the service group process is inactive.

   `F`

   Indicates that the service group process is specified as the target of the server UAP simulator (and cannot be activated or inactivated).

6. Debugger connection:

   `D`

   Specified

   `*`

   Not specified

7. Debugger process ID

   `*****` is displayed when the process is inactive.

13.2.5 read (input of tester file name to offline tester)

(1) Syntax

`read` *tester-file-name* `[+` *MCF-receive-message-file-name*`]`

(2) Function

Informs the offline tester of the tester file name required by a simulator.

(3) Command arguments

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

  Specify the name of the tester file required by the offline tester.

  The prompt displays which tester file name is required, as shown below.

  Format

```
read(group1:service1:rpc_rtn)?>
         1.        2.        3.
```

1. Service group name
2. Service name (not displayed for a process other than a service)
3. Tester file type:

`rpc_rtn`

Service response data file

`crm_rtn`

XATMI response data file

`trp_trn`

TxRPC response data file

`crm_rcv`

XATMI receive data file

`mcf_msg`

MCF receive message file

`adm_cmd`

Operating command result data file

- *MCF-receive-message-file-name* ~<pathname>

  When concatenating the tester file with an MCF receive message file, write a plus sign (+), then specify the name of the MCF receive message file.

13.2.6 start (service group activation)

(1) Syntax

```
start {SPP|MHP} service-group-name
```

(2) Function

Reactivates a UAP when:

- Suppression of service group activation is specified for the UAP at offline tester startup
- The UAP terminates abnormally during testing

(3) Command arguments

■ `SPP|MHP`

Specify the type of service group to be activated.

`SPP`

Indicates an SPP.

`MHP`

Indicates an MHP.

■ *service-group-name* ~<identifier of 1-31 characters>

Specify the name of the service group to be activated.

The service group name must be defined in the offline tester environment definition.

13.2.7 stop (service group termination)

(1) Syntax

```
stop {SPP|MHP} service-group-name
```

(2) Function

Terminates an active UAP.

(3) Command arguments

■ `SPP|MHP`

Specify the type of service group to be terminated.

`SPP`

Indicates an SPP.

`MHP`

Indicates an MHP.

■ *service-group-name* ~<identifier of 1-31 characters>

Specify the name of the service group to be terminated.

The service group name must be defined in the offline tester environment definition.

13.2.8 write (input of tester file name to offline tester)

(1) Syntax

```
write tester-file-name
```

(2) Function

Informs the offline tester of the tester file name required by a simulator.

(3) Command argument

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

  Specify the name of the tester file required by the offline tester.

  The prompt displays which tester file name is required, as shown below.

  Format

$$\texttt{write(}\underbrace{\texttt{group1}}_{1.}\texttt{:}\underbrace{\texttt{service1}}_{2.}\texttt{:}\underbrace{\texttt{crm\_snd}}_{3.}\texttt{)?>}$$

1. Service group name
2. Service name (not displayed for a process other than a service)
3. Tester file type:

`crm_snd`

XATMI send data file

Chapter

14. Simulation Functions

This chapter describes the purpose, processing, and return values of the simulation functions provided by the offline tester.

This chapter contains the following sections:

14.1 List of simulation functions and processing
14.2 List of return values for simulation functions
14.3 List of functions not supported by the simulation feature

14.1 List of simulation functions and processing

This section lists the offline tester simulation functions and provides notes on function simulations.

(1) Simulation functions

Table 14-1 lists the offline tester simulation functions for simulating OpenTP1 functions.

Table 14-1: List of offline tester simulation functions

Type	Function name [prog_name (request_code)] <DML>	Purpose	Traces	Return value	Function processing
Control of system operation (adm)	dc_adm_call_command [CBLDCADM(COMMAND)]	Executes an operating command.	Y	Y	Returns data from the operating command result data file.
	dc_adm_complete [CBLDCADM(COMPLETE)]	Notifies completion of user server startup.	Y	Y	--
	dc_adm_status [CBLDCADM(STATUS)]	Notifies user server status.	Y	Y	Returns DCADM_STAT_START_NORMAL (return value) or zero (return code) at normal termination.
	dc_adm_get_nd_status_begin	Starts status acquisition at the OpenTP1 node.	Y	Y	Gets the number of node IDs set in the function return values file.
	dc_adm_get_nd_status_next	Gets OpenTP1 node status.	Y	Y	Gets the node ID set in the function return values file. Returns DCADM_STATUS_NORMAL (C return value) at normal termination.
	dc_adm_get_nd_status_done	Ends status acquisition at the OpenTP1 node.	Y	Y	--

Type	Function name [prog_name (request_code)] <DML>	Purpose	Traces	Return value	Function processing
	dc_adm_get_nd_status	Gets OpenTP1 node status.	Y	Y	Returns DCADM_STATUS_NORMAL (return value) at normal termination.
	dc_adm_get_node_id	Gets the local node ID from the system common definition.	Y	Y	Gets the node ID set in the function return values file.
	dc_adm_get_sv_status_begin	Starts server status acquisition.	Y	Y	Gets the number of server names set in the function return values file.
	dc_adm_get_sv_status_next	Gets server status at the OpenTP1 node.	Y	Y	Gets the server name set in the function return values file. Returns DCADM_STATUS_NORMAL (C return value) at normal termination.
	dc_adm_get_sv_status_done	Ends server status acquisition.	Y	Y	--
	dc_adm_get_sv_status	Gets status of a specified server.	Y	Y	Returns DCADM_STATUS_NORMAL (return value) at normal termination.
	dc_adm_get_nodeconf_begin	Starts node ID acquisition.	Y	Y	Returns the number of node IDs set in function return values file.
	dc_adm_get_nodeconf_next	Gets multi-node area ID for the UAP that issued the function, or all node IDs of specified subareas.	Y	Y	Returns the node IDs set in the function return values file.

Type	Function name [prog_name (request_code)] <DML>	Purpose	Traces	Return value	Function processing
	dc_adm_get_nodeconf_done	Ends node ID acquisition.	Y	Y	--
DAM file service (dam)	dc_dam_close [CBLDCDAM(CLOS)]	Closes a DAM file.	Y	Y	Closes a DAM file.
	dc_dam_create [CBLDCDMB(CRAT)]	Allocates a physical file.	N	N	Creates a DAM file and returns the file descriptor.
	dc_dam_end [CBLDCDAM(END)]	Declares to stop using files not subject to recovery.	Y	Y	--
	dc_dam_get [CBLDCDMB(GET)]	Reads a physical file block.	N	N	Reads a specified block from a DAM file to a specified buffer.
	dc_dam_hold [CBLDCDAM(HOLD)]	Logical shutdown of a DAM file	Y	Y	Sets shutdown status in the DAM file header and shuts down the DAM file.
	dc_dam_iclose [CBLDCDMB(CLOS)]	Closes a physical file.	N	N	Closes a DAM file.
	dc_dam_iopen [CBLDCDMB(OPEN)]	Opens a physical file.	N	N	Opens a DAM file and returns the file descriptor.
	dc_dam_open [CBLDCDAM(OPEN)]	Opens a DAM file.	Y	Y	Opens a DAM file and returns the file descriptor. Locks the file if lock is specified for the file.
	dc_dam_put [CBLDCDMB(PUT)]	Writes a physical file block.	N	N	Writes buffer contents to a specified DAM file block.

Type	Function name [prog_name (request_code)] <DML>	Purpose	Traces	Return value	Function processing
	`dc_dam_read` `[CBLDCDAM(READ)]`	Reads a DAM file block.	Y	Y	Reads a specified DAM file block to a specified buffer. Locks the file if lock is specified for the block.
	`dc_dam_start` `[CBLDCDAM(STRT)]`	Declares to start using files not subject to recovery.	Y	Y	--
	`dc_dam_status` `[CBLDCDAM(STAT)]`	Shows DAM file state.	Y	Y	Returns the DAM file state.
	`dc_dam_release` `[CBLDCDAM(RLSE)]`	Releases DAM file shutdown status.	Y	Y	Resets the shutdown status in the DAM file header and cancels the shutdown of the DAM file.
	`dc_dam_rewrite` `[CBLDCDAM(REWT)]`	Updates a DAM file block.	Y	Y	Writes the contents of a specified buffer to a specified DAM file block.
	`dc_dam_write` `[CBLDCDAM(WRIT)]`	Outputs a DAM file.	Y	Y	Writes the contents of a specified buffer to a specified DAM file block.
Shared table service (`ist`)	`dc_ist_close` `[CBLDCIST(CLOS)]`	Closes IST table.	Y	Y	Closes the IST table.
	`dc_ist_open` `[CBLDCIST(OPEN)]`	Opens IST table.	Y	Y	Opens the IST table and returns its descriptor.
	`dc_ist_read` `[CBLDCIST(READ)]`	Reads records from IST table.	Y	Y	Reads specified records from the IST table to specified buffer.

Type	Function name [prog_name (request_code)] <DML>	Purpose	Traces	Return value	Function processing
	`dc_ist_write` [CBLDCIST(WRIT)]	Writes records to IST table.	Y	Y	Writes specified records to the IST table.
User journal collection (jnl)	`dc_jnl_ujput` [CBLDCJNL(UJPUT)]	Collects UAP log information.	Y	Y	--
Lock of resources (lck)	`dc_lck_get` [CBLDCLCK(GET)]	Requests locking of resources.	Y	Y	--
	`dc_lck_release_all` [CBLDCLCK(RELALL)]	Requests unlocking of all resources.	Y	Y	--
	`dc_lck_release_byname` [CBLDCLCK(RELNAME)]	Requests unlocking of a specified resource.	Y	Y	--
Message log control (log)	`dc_logprint` [CBLDCLOG(PRINT)]	Requests logged message output.	Y	Y	--
Message control function (mcf)	`dc_mcf_execap` [CBLDCMCF(EXECAP)] <SEND>	Starts an application.	Y	Y	--
	`dc_mcf_mainloop` [CBLDCMCF(MAINLOOP)]	Starts the MCF service.	Y	Y	Notifies the offline tester that MCF service has started. At a service request to the MHP, executes the service function and waits for the next service request. Returns when a UAP termination request is received (at offline tester termination, for example).

Type	Function name [prog_name (request_code)] <DML>	Purpose	Traces	Return value	Function processing
	`dc_mcf_receive` `[CBLDCMCF(RECEIVE)]<RECEIVE>`	Message receive	Y	Y	Inputs a segment from the MCF receive message file and stores the segment in the message receive area. Counts up the transaction sequence number.
	`dc_mcf_reply` `[CBLDCMCF(REPLY)]` `<SEND>`	Response message send	Y	Y	--
	`dc_mcf_rollback` `[CBLDCMCF(ROLLBACK)]` `<ROLLBACK>`	Partial recovery	Y	Y	Counts up the transaction sequence number if the next processing is specified to run as a different transaction.
	`dc_mcf_send` `[CBLDCMCF(SEND)]` `<SEND>`	Message send	Y	Y	--
	`dc_mcf_open` `[CBLDCMCF(OPEN)]`	Prepares and initializes for using the MCF service.	Y	Y	--
	`dc_mcf_close` `[CBLDCMCF(CLOSE)]`	Deletes the environment for using the MCF service.	Y	N	--
	`dc_mcf_sendrecv` `[CBLDCMCF(SENDRECV)]` `<SEND>`	Synchronous message send/ receive	Y	Y	Outputs trace information of the last segment, then inputs a segment from the MCF receive message file and stores the segment in the message receive area.

Type	Function name [prog_name (request_code)] <DML>	Purpose	Traces	Return value	Function processing
	dc_mcf_recvsync [CBLDCMCF(RECVSYNC)] <RECEIVE>	Synchronous message receive	Y	Y	Inputs a segment from the MCF receive message file and stores the segment in the message receive area.
	dc_mcf_sendsync [CBLDCMCF(SENDSYNC)] <SEND>/<ENABLE>/ <DISABLE>	Synchronous message send	Y	Y	--
	dc_mcf_tempget [CBLDCMCF(TEMPGET)] <RECEIVE>	Passes temporary memory data for continuous inquiry/response	Y	Y	Inputs data from the temporary memory data file and stores the data in the message receive area. Or, stores a null character if no file exists.
	dc_mcf_tempput [CBLDCMCF(TEMPPUT)] <SEND>	Updates temporary memory data for continuous inquiry/response	Y	Y	Updates the temporary memory data file. Or, creates an update file if none exists.
	dc_mcf_contend [CBLDCMCF(CONTEND)] <DISABLE>	Terminates continuous inquiry/response	Y	Y	Deletes the temporary memory data file.
	dc_mcf_regster	Sets user exit routine function addresses.	Y	Y	--
	dc_mcf_resend [CBLDCMCF(RESEND)]	Message resend	Y	Y	--
	dc_mcf_commit [CBLDCMCF(COMMIT)]	Synchronous point acquisition	Y	Y	Counts up the transaction sequence number.

Type	Function name [prog_name (request_code)] <DML>	Purpose	Traces	Return value	Function processing
Remote procedure call (`rpc`)	`dc_rpc_call` `[CBLDCRPC(CALL)]`	Remote service call	Y	Y	Requests the offline tester to execute a service function. Returns a descriptor (positive integer) as the return value when `DCRPC_NOWAIT` is specified. Or, returns zero to the specified service (service function) as the response length when `DCRPC_NOREPLY` is specified.
	`dc_rpc_close` `[CBLDCRPC(CLOSE)]`	UAP termination	Y	N	--
	`dc_rpc_mainloop` `[CBLDCRSV(MAINLOOP)]`	Starts the SPP service.	Y	Y	Notifies the offline tester that service has started. At a service request to the SPP, executes the service function and waits for the next service request. Returns when a UAP termination request is received (at offline tester termination, for example).
	`dc_rpc_open` `[CBLDCRPC(OPEN)]`	UAP start processing	Y	Y	Allocates shared memory, then notifies the offline tester that the UAPs have started.

Type	Function name [prog_name (request_code)] <DML>	Purpose	Traces	Return value	Function processing
	dc_rpc_poll_any_replies [CBLDCRPC(POLLANYR)]	Receives responses from the dc_rpc_call function (DCRPC_NOWAIT specified).	Y	Y	If flags=DCNOFLAGS, returns the descriptor of the first dc_rpc_call function (DCRPC_NOWAIT specified) for which no reply was received. If flags=DCRPC_SPECIFIC_MSG, returns DC_OK. If no dc_rpc_call functions that terminated normally were issued in the SPP, returns DCRPC_PROTO.
	dc_rpc_discard_further_replies [CBLDCRPC(DISCARDF)]	Cancels responses from the dc_rpc_call function (DCRPC_NOWAIT specified).	Y	N	Cancels all descriptors returned by the dc_rpc_call function (DCRPC_NOWAIT specified).
	dc_rpc_get_callers_address [CBLDCRPC(GETCLADR)]	Notifies the node address of the client.	Y	N	Returns ADDRESS (fixed value) as the client address.
	dc_rpc_set_service_prio [CBLDCRPC(SETSVPRI)]	Sets schedule priority of service requests.	Y	N	--
	dc_rpc_get_service_prio [CBLDCRPC(GETSVPRI)]	Gets schedule priority of service requests.	Y	N	Returns the schedule priority value specified for the dc_rpc_set_service_prio function.

Type	Function name [prog_name (request_code)] <DML>	Purpose	Traces	Return value	Function processing
	dc_rpc_set_watch_time [CBLDCRPC(SETWATCH)]	Updates the service response wait time.	Y	Y	Updates the service response wait time.
	dc_rpc_get_watch_time [CBLDCRPC(GETWATCH)]	References the service response wait time.	Y	N	References the values set by the dc_rpc_set_watch_time function. Returns 180 if the function has not been issued.
TAM file service (tam)	dc_tam_close	Closes a TAM table.	Y	Y	Releases lock and closes the TAM table.
	dc_tam_delete [CBLDCTAM(ERS or ERSR)]	Deletes a record from a TAM table.	Y	Y	Deletes a record specified by key value from a TAM table and updates the TAM table file.
	dc_tam_get_inf [CBLDCTAM(GST)]	Collects TAM table information.	Y	Y	Returns DCTAM_STS_OPN if the calling process has issued an open request for the specified TAM table file. Or, returns DCTAM_STS_CLS if no open request has been issued.
	dc_tam_open	Opens a TAM table.	Y	Y	Opens the TAM table specified by table ID and returns the file ID as the table ID. Locks the TAM table file if lock of the TAM table is specified.

Type	Function name [prog_name (request_code)] <DML>	Purpose	Traces	Return value	Function processing
	`dc_tam_read` [`CBLDCTAM(FxxR` or `FxxU)`]	Retrieves a record from a TAM table.	Y	Y	Retrieves a specified index from a TAM table (control part and index part) in shared memory and reads the record for the index from the TAM table file. Locks the TAM table file if lock of the record is specified.
	`dc_tam_read_cancel`	Cancels TAM table record retrieval.	Y	Y	Unlocks the TAM table file that contains a specified record.
	`dc_tam_rewrite`	Updates a retrievable record in a TAM table.	Y	Y	Writes the contents of a specified buffer to a specified record in a TAM table.
	`dc_tam_write` [`CBLDCTAM(MFY`, `MFYS`, or `STR)`]	Updates or appends a record in a TAM table.	Y	Y	Retrieves a specified index from a TAM table (control part and index part) in shared memory and writes the contents of a specified buffer to the record for the index in the TAM table file.
Transaction control (`trn`)	`dc_trn_begin` [`CBLDCTRN(BEGIN)`]	Starts a transaction.	Y	Y	Counts up the transaction sequence number.
	`dc_trn_chained_commit` [`CBLDCTRN(C-COMMIT)`]	Commits a transaction (chained mode).	Y	Y	Counts up the transaction sequence number.

Type	Function name [prog_name (request_code)] <DML>	Purpose	Traces	Return value	Function processing
	dc_trn_chained_rollback [CBLDCTRN(C-ROLL)]	Rolls back a transaction (chained mode).	Y	Y	Counts up the transaction sequence number.
	dc_trn_info [CBLDCTRN(INFO)]	Returns information for the current transaction.	Y	Y	Returns zero if no information is specified in the function return values file.
	dc_trn_unchained_commit [CBLDCTRN(U-COMMIT)]	Commits a transaction (unchained mode).	Y	Y	--
	dc_trn_unchained_rollback [CBLDCTRN(U-ROLL)]	Rolls back a transaction (unchained mode).	Y	Y	--
TX interface (tx_~)	tx_begin [TXBEGIN]	Starts a transaction.	Y	Y	Counts up the transaction sequence number and initializes TXINFO information.
	tx_close [TXCLOSE]	Closes the resource managers.	Y	Y	--
	tx_commit [TXCOMMIT]	Commits a transaction.	Y	Y	In chained mode, counts up the transaction sequence number.
	tx_info [TXINFORM]	Returns information for the current transaction.	Y	Y	Returns zero if no information is specified in the function return values file.
	tx_open [TXOPEN]	Opens the resource managers.	Y	Y	--
	tx_set_commit_return [TXSETCOMMITRET]	Sets commit_return characteristics.	Y	Y	--

Type	Function name [prog_name (request_code)] <DML>	Purpose	Traces	Return value	Function processing
	tx_set_transaction_control [TXSETTRANCTL]	Sets trans-action_control characteristics.	Y	Y	Sets transaction_control characteristics.
	tx_set_transaction_timeout [TXSETTIMEOUT]	Sets trans-action_timeout characteristics.	Y	Y	--
	tx_rollback [TXROLLBACK]	Rolls back a transaction.	Y	Y	In chained mode, counts up the transaction sequence number and sets transaction_state characteristics.
XATMI interface (tp_~)	tpalloc	Allocates a typed buffer.	Y	Y	Allocates the buffer specified by an argument of type type and returns the pointer.
	tpfree	Frees a typed buffer.	Y	N	Frees the buffer allocated by the tpalloc or tprealloc function.
	tprealloc	Resizes a typed buffer.	Y	Y	Resizes the buffer allocated by the tpalloc or tprealloc function.
	tptypes	Gets typed buffer information.	Y	Y	Returns the type and subtype of the buffer allocated by the tpalloc or tprealloc function.

Type	Function name [prog_name (request_code)] <DML>	Purpose	Traces	Return value	Function processing
	`tpservice`	Service function template	Y	N	Collects trace information immediately before a service function is called.
	`tpreturn`	Returns from a service function.	Y	Y	Sets return information and returns to the client UAP.
	`tpadvertise`	Advertises a service name.	Y	Y	--
	`tpunadvertise`	Cancels service name advertising.	Y	Y	--
	`tpacall`	Asynchronous service request	Y	Y	Requests the offline tester to execute a service function. The `tpgetrply` function returns the call result.
	`tpcall`	Synchronous service request	Y	Y	Requests the offline tester to execute a service function.
	`tpcancel`	Service cancellation	Y	Y	Cancels the response from the service requested by `tpacall` function.
	`tpgetrply`	Asynchronous response from a service	Y	Y	Returns the execution result of a service function.
	`tpconnect`	Establishes the conversational service paradigm connection.	Y	Y	Requests the offline tester to execute a service function. The execution result is returned by the `tprecv` function.

Type	Function name [prog_name (request_code)] <DML>	Purpose	Traces	Return value	Function processing
	`tpdiscon`	Disconnects the conversational service paradigm.	Y	Y	Terminates the service if in reply wait state (`tprecv` function) and disables acceptance of `tpsend` or `tprecv` after the `tpdiscon` is accepted.
	`tprecv`	Message receive from the conversational service paradigm	Y	Y	Inputs data from the XATMI receive data file.
	`tpsend`	Message send to the conversational service paradigm	Y	Y	Outputs data to the XATMI send data file.
Online tester (`uto`)	`dc_uto_test_status` [`CBLDCUTO(T-STATUS)`]	Reports user server test state.	Y	Y	Returns non-test mode state.

Legend:

Y: Trace information collected; return value set.

N: Trace information cannot be collected; return value cannot be set.

--: No processing

(2) Notes on simulation functions

Note the following points on using the function simulator:

1. The offline tester does not check the type of the UAP issuing the function, transaction status, or whether the function is issued inside or outside the `main` function.
2. The function sequence is checked only for functions that affect offline tester operation.
3. Arguments are not checked. The user should check the arguments from the trace information.
4. An error message is output but no trace information is collected when an interface code or request code is set incorrectly in a COBOL program.

5. A dc_trn_~ function cannot coexist with a tx_~ function. The offline tester does not check whether the two functions types are mixed.

6. The offline tester counts the number of transactions (transaction sequence number). The transaction sequence number is counted up at execution of some simulation functions and at execution of the call subcommand. You can reference the transaction sequence number by using the tx_info function. For details about the simulation function that increments the transaction sequence number, see *(1) Simulation functions*, above.

14.2 List of return values for simulation functions

Table 14-2 lists the return values for simulation functions. Note that `0`, `DC_OK`, `DCMCFRTN_00000`, and `TX_OK` are omitted.

Table 14-2: List of return values for simulation functions

Type	Function name [prog_name (request_code)] <DML>	C return value	COBOL return code
Control of system operation (adm)	dc_adm_call_command [CBLDCADM(COMMAND)]	DCADMER_STATNOTZERO DCADMER_PARAM DCADMER_MEMORY_OUT DCADMER_MEMORY_ERR DCADMER_MEMORY_OUTERR DCADMER_PROTO	01801 01802 01803 01804 01805 01807[#1]
	dc_adm_complete [CBLDCADM(COMPLETE)]	DCADM_STAT_START_NORMAL DCADMER_PROTO DCADMER_PARAM	00000 01830[#1] 01831
	dc_adm_status [CBLDCADM(STATUS)]	DCADMER_PROTO DCADMER_PARAM	01830[#1] 01831
	dc_adm_get_nd_status_begin	DCADMER_PROTO DCADMER_PARAM	--[#1, #2] --
	dc_adm_get_nd_status_next	DCADM_STAT_START_NORMAL DCADMER_PROTO DCADMER_PARAM DCADMER_NO_MORE_ENTRY	-- --[#1, #3] -- --
	dc_adm_get_nd_status_done	DCADMER_PROTO DCADMER_PARAM	--[#1, #3] --
	dc_adm_get_nd_status	DCADM_STAT_START_NORMAL DCADMER_PROTO DCADMER_PARAM	-- --[#1, #2] --
	dc_adm_get_node_id	DCADMER_PROTO DCADMER_PARAM	--[#1, #2] --
	dc_adm_get_sv_status_begin	DCADMER_PROTO DCADMER_PARAM	--[#1, #2] --

Type	Function name [prog_name (request_code)] <DML>	C return value	COBOL return code
	dc_adm_get_sv_status_next	DCADM_STAT_START_NORMAL DCADMER_PROTO DCADMER_PARAM DCADMER_NO_MORE_ENTRY	-- --#1, #3 -- --
	dc_adm_get_sv_status_done	DCADMER_PROTO DCADMER_PARAM	--#1, #3 --
	dc_adm_get_sv_status	DCADM_STAT_START_NORMAL DCADMER_PROTO DCADMER_PARAM	-- --#1, #2 --
	dc_adm_get_nodeconf_begin	DCADMER_PROTO DCADMER_PARAM	--#1, #2 --
	dc_adm_get_nodeconf_next	DCADMER_PROTO DCADMER_PARAM DCADMER_NO_MORE_ENTRY	--#1, #3 -- --
	dc_adm_get_nodeconf_done	DCADMER_PROTO DCADMER_PARAM	--#1, #3 --
DAM file service (dam)	dc_dam_close [CBLDCDAM(CLOS)]	DCDAMER_PROTO DCDAMER_BADF DCDAMER_PARAM_FLAGS	01600#1 01603 01611
	dc_dam_create [CBLDCDMB(CRAT)]	DCDAMER_NOMEM DCDAMER_OPENED DCDAMER_PARAM_FLAGS DCDAMER_FILEER DCDAMER_PNUMER DCDAMER_EXIST DCDAMER_IOER DCDAMER_OPENNUM DCDAMER_ACCESS DCDAMER_LBLNER DCDAMER_LBNOER DCDAMER_LFNOVF	01607 01608 01611 01614 01615 01617 01620 01627 01628 01630 01631 01635
	dc_dam_end [CBLDCDAM(END)]	DCDAMER_PROTO DCDAMER_PARAM_FLAGS	01600#1 01611

Type	Function name [prog_name (request_code)] <DML>	C return value	COBOL return code
	dc_dam_get [CBLDCDMB(GET)]	DCDAMER_BADF DCDAMER_BUFER DCDAMER_SEQER DCDAMER_PARAM_FLAGS DCDAMER_IOER DCDAMER_EOF	01603 01604 01605 01611 01620 01637
	dc_dam_hold [CBLDCDAM(HOLD)]	DCDAMER_PROTO DCDAMER_UNDEF DCDAMER_PARAM_LFNAME DCDAMER_PARAM_FLAGS DCDAMER_IOER DCDAMER_LHOLDED DCDAMER_OHOLDED	01600[#1] 01601 01610 01611 01620 01625 01626
	dc_dam_iclose [CBLDCDMB(CLOS)]	DCDAMER_BADF DCDAMER_PARAM_FLAGS	01603 01611
	dc_dam_iopen [CBLDCDMB(OPEN)]	DCDAMER_NOMEM DCDAMER_OPENED DCDAMER_PARAM_FLAGS DCDAMER_FILEER DCDAMER_PNUMER DCDAMER_NOEXIST DCDAMER_IOER DCDAMER_OPENNUM DCDAMER_ACCESS DCDAMER_LFNOVF	01607 01608 01611 01614 01615 01619 01620 01627 01628 01635
	dc_dam_open [CBLDCDAM(OPEN)]	DCDAMER_PROTO DCDAMER_UNDEF DCDAMER_EXCER DCDAMER_OPENED DCDAMER_PARAM_LFNAME DCDAMER_PARAM_FLAGS DCDAMER_IOER DCDAMER_LHOLD DCDAMER_OHOLD DCDAMER_OPENNUM DCDAMER_ACCESS	01600[#1] 01601 01602 01608 01610 01611 01620 01621 01622 01627 01628

Type	Function name [prog_name (request_code)] <DML>	C return value	COBOL return code
	dc_dam_put [CBLDCDMB(PUT)]	DCDAMER_BADF DCDAMER_BUFER DCDAMER_SEQER DCDAMER_PARAM_FLAGS DCDAMER_IOER DCDAMER_EOF	01603 01604 01605 01611 01620 01637
	dc_dam_read [CBLDCDAM(READ)]	DCDAMER_PROTO DCDAMER_EXCER DCDAMER_BADF DCDAMER_BUFER DCDAMER_BNOER DCDAMER_PARAM_KEYNO DCDAMER_PARAM_FLAGS DCDAMER_IOER DCDAMER_LHOLD DCDAMER_OHOLD	01600[#1] 01602 01603 01604 01606 01609 01611 01620 01621 01622
	dc_dam_start [CBLDCDAM(STRT)]	DCDAMER_PROTO DCDAMER_PARAM_FLAGS DCDAMER_STARTED	01600[#1] 01611 01647
	dc_dam_status [CBLDCDAM(STAT)]	DCDAMER_PROTO DCDAMER_UNDEF DCDAMER_PARAM_LFNAME DCDAMER_PARAM_FLAGS DCDAMER_PARAM_ERROR DCDAMER_IOER	01600[#1] 01601 01610 01611 01612 01620
	dc_dam_release [CBLDCDAM(RLSE)]	DCDAMER_PROTO DCDAMER_UNDEF DCDAMER_PARAM_LFNAME DCDAMER_PARAM_FLAGS DCDAMER_IOER DCDAMER_NOLHOLD DCDAMER_NOOHOLD	01600[#1] 01601 01610 01611 01620 01623 01624

Type	Function name [prog_name (request_code)] <DML>	C return value	COBOL return code
	dc_dam_rewrite [CBLDCDAM(REWT)]	DCDAMER_PROTO DCDAMER_BADF DCDAMER_BUFER DCDAMER_BNOER DCDAMER_PARAM_KEYNO DCDAMER_PARAM_FLAGS DCDAMER_IOER DCDAMER_LHOLD DCDAMER_OHOLD DCDAMER_BUFOV	01600[#1] 01603 01604 01606 01609 01611 01620 01621 01622 01641
	dc_dam_write [CBLDCDAM(WRIT)]	DCDAMER_PROTO DCDAMER_EXCER DCDAMER_BADF DCDAMER_BUFER DCDAMER_BNOER DCDAMER_PARAM_KEYNO DCDAMER_PARAM_FLAGS DCDAMER_IOER DCDAMER_LHOLD DCDAMER_OHOLD DCDAMER_BUFOV	01600[#1] 01602 01603 01604 01606 01609 01611 01620 01621 01622 01641
Shared table service (ist)	dc_ist_close [CBLDCIST(CLOS)]	DCISTER_PROTO DCISTER_BADID DCISTER_PARAM_FLAGS	--[#1] -- --
	dc_ist_open [CBLDCIST(OPEN)]	DCISTER_PROTO DCISTER_UNDEF DCISTER_OPENED DCISTER_PARAM_TBLNAME DCISTER_PARAM_FLAGS	--[#1] -- -- -- --
	dc_ist_read [CBLDCIST(READ)]	DCISTER_PROTO DCISTER_BADID DCISTER_BUFER DCISTER_RNOER DCISTER_NOMEM DCISTER_PARAM_KEYNO DCISTER_PARAM_FLAGS	--[#1] -- -- -- -- -- --

Type	Function name [prog_name (request_code)] <DML>	C return value	COBOL return code
	dc_ist_write [CBLDCIST(WRIT)]	DCISTER_PROTO DCISTER_BADID DCISTER_BUFER DCISTER_RNOER DCISTER_NOMEM DCISTER_PARAM_KEYNO DCISTER_PARAM_FLAGS DCISTER_BUFOV	--[#1] -- -- -- -- -- -- --
User journal collection (jnl)	dc_jnl_ujput [CBLDCJNL(UJPUT)]	DCJNLER_PARAM DCJNLER_SHORT DCJNLER_PROTO	01101 01102 01105[#1]
Lock of resources (lck)	dc_lck_get [CBLDCLCK(GET)]	DCLCKER_PARAM DCLCKER_OUTOFTRN	00401 00455[#1]
	dc_lck_release_all [CBLDCLCK(RELALL)]	DCLCKER_PARAM DCLCKER_OUTOFTRN	00401 00455[#1]
	dc_lck_release_byname [CBLDCLCK(RELNAME)]	DCLCKER_PARAM DCLCKER_OUTOFTRN	00401 00455[#1]
Message log control (log)	dc_logprint [CBLDCLOG(PRINT)]	DCLOGER_PARAM_ARGS DCLOGER_COMM	01900 01901[#1]
Message control function (mcf)	dc_mcf_execap [CBLDCMCF(EXECAP)] <SEND>	DCMCFER_PROTO DCMCFRTN_71002 DCMCFRTN_72000 DCMCFRTN_72001 DCMCFRTN_72005 DCMCFRTN_72016 DCMCFRTN_72024 DCMCFRTN_72026 DCMCFRTN_72041 DCMCFRTN_72108	70901[#1, #4] 71002 72000 72001 72005 72016 72024 72026 72041 72108
	dc_mcf_mainloop [CBLDCMCF(MAINLOOP)]	DCMCFER_INVALID_ARGS DCMCFER_PROTO DCMCFER_FATAL	70900 70901[#1, #5] 70902

Type	Function name [prog_name (request_code)] <DML>	C return value	COBOL return code
	dc_mcf_receive [CBLDCMCF(RECEIVE)] <RECEIVE>	DCMCFER_PROTO DCMCFRTN_71000 DCMCFRTN_71001 DCMCFRTN_71002 DCMCFRTN_72000 DCMCFRTN_72001 DCMCFRTN_72013 DCMCFRTN_72016 DCMCFRTN_72024 DCMCFRTN_72025 DCMCFRTN_72036	70901[#1, #4] 71000 71001 71002 72000 72001 72013 72016 72024 72025 72036
	dc_mcf_reply [CBLDCMCF(REPLY)] <SEND>	DCMCFER_PROTO DCMCFRTN_71002 DCMCFRTN_72000 DCMCFRTN_72005 DCMCFRTN_72016 DCMCFRTN_72017 DCMCFRTN_72026 DCMCFRTN_72041 DCMCFRTN_72047	70901[#1, #4] 71002 72000 72005 72016 72017 72026 72041 72047
	dc_mcf_rollback [CBLDCMCF(ROLLBACK)] <ROLLBACK>	DCMCFER_PROTO DCMCFRTN_72000 DCMCFRTN_72027	70901[#1, #4] 72000 72027
	dc_mcf_send [CBLDCMCF(SEND)] <SEND>	DCMCFER_PROTO DCMCFRTN_71002 DCMCFRTN_72000 DCMCFRTN_72001 DCMCFRTN_72005 DCMCFRTN_72016 DCMCFRTN_72017 DCMCFRTN_72020 DCMCFRTN_72024 DCMCFRTN_72026 DCMCFRTN_72041	70901[#1, #4] 71002 72000 72001 72005 72016 72017 72020 72024 72026 72041
	dc_mcf_open [CBLDCMCF(OPEN)]	DCMCFER_INVALID_ARGS DCMCFER_PROTO	70900 70901[#1]
	dc_mcf_close [CBLDCMCF(CLOSE)]	--	--

Type	Function name [prog_name (request_code)] <DML>	C return value	COBOL return code
	dc_mcf_sendrecv [CBLDCMCF(SENDRECV)] <SEND>	DCMCFER_PROTO DCMCFRTN_71002 DCMCFRTN_71108 DCMCFRTN_72000 DCMCFRTN_72001 DCMCFRTN_72005 DCMCFRTN_72013 DCMCFRTN_72016 DCMCFRTN_72024 DCMCFRTN_72026 DCMCFRTN_72036 DCMCFRTN_72041	70901[#1, #4] 71002 71108 72000 72001 72005 72013 72016 72024 72026 72036 72041
	dc_mcf_recvsync [CBLDCMCF(RECVSYNC)] <RECEIVE>	DCMCFER_PROTO DCMCFRTN_71001 DCMCFRTN_71108 DCMCFRTN_72000 DCMCFRTN_72001 DCMCFRTN_72013 DCMCFRTN_72016 DCMCFRTN_72024 DCMCFRTN_72025 DCMCFRTN_72036 DCMCFRTN_73001	70901[#1, #4] 71001 71108 72000 72001 72013 72016 72024 72025 72036 73001
	dc_mcf_sendsync [CBLDCMCF(SENDSYNC)] <SEND>/<ENABLE>/<DISABLE>	DCMCFER_PROTO DCMCFRTN_71002 DCMCFRTN_72000 DCMCFRTN_72001 DCMCFRTN_72005 DCMCFRTN_72016 DCMCFRTN_72024 DCMCFRTN_72026 DCMCFRTN_72041	70901[#1, #4] 71002 72000 72001 72005 72016 72024 72026 72041
	dc_mcf_tempget [CBLDCMCF(TEMPGET)] <RECEIVE>	DCMCFER_PROTO DCMCFRTN_72000 DCMCFRTN_72013 DCMCFRTN_72016 DCMCFRTN_72036 DCMCFRTN_72106	70901[#1, #4] 72000 72013 72016 72036 72106

Type	Function name [prog_name (request_code)] <DML>	C return value	COBOL return code
	dc_mcf_tempput [CBLDCMCF(TEMPPUT)] <SEND>	DCMCFER_PROTO DCMCFRTN_71103 DCMCFRTN_72000 DCMCFRTN_72013 DCMCFRTN_72016 DCMCFRTN_72035 DCMCFRTN_72106	70901[#1, #4] 71103 72000 72013 72016 72035 72106
	dc_mcf_contend [CBLDCMCF(CONTEND)] <DISABLE>	DCMCFER_PROTO DCMCFRTN_72000 DCMCFRTN_72016	70901[#1] 72000 72016
	dc_mcf_regster	DCMCFER_INVALID_ARGS DCMCFER_PROTO	-- --[#1]
	dc_mcf_resend [CBLDCMCF(RESEND)]	DCMCFER_PROTO DCMCFRTN_72000 DCMCFRTN_72001 DCMCFRTN_72011 DCMCFRTN_72016 DCMCFRTN_72017 DCMCFRTN_72024 DCMCFRTN_72047	70901[#1, #4] 72000 72001 72011 72016 72017 72024 72047
	dc_mcf_commit [CBLDCMCF(COMMIT)]	DCMCFER_PROTO DCMCFRTN_72000 DCMCFRTN_72016	70901[#1, #4] 72000 72016
Remote procedure call (rpc)	dc_rpc_call [CBLDCRPC(CALL)]	DCRPCER_INVALID_ARGS DCRPCER_PROTO DCRPCER_MESSAGE_TOO_BIG DCRPCER_REPLY_TOO_BIG DCRPCER_NO_SUCH_SERVICE_GROUP DCRPCER_NO_SUCH_SERVICE DCRPCER_SERVICE_CLOSED DCRPCER_SYSERR_AT_SERVER DCRPCER_SYSER	00301 00302[#1, #5] 00308 00309 00310 00311 00312 00316 00318
	dc_rpc_close [CBLDCRPC(CLOSE)]	--	--
	dc_rpc_mainloop [CBLDCRSV(MAINLOOP)]	DCRPCER_INVALID_ARGS DCRPCER_PROTO DCRPCER_FATAL	00301 00302[#1, #5] 00303

Type	Function name [prog_name (request_code)] <DML>	C return value	COBOL return code
	dc_rpc_open [CBLDCRPC(OPEN)]	DCRPCER_INVALID_ARGS DCRPCER_PROTO DCRPCER_FATAL	00301 00302 00303
	dc_rpc_poll_any_replies [CBLDCRPC(POLLANYR)]	DCRPCER_INVALID_ARGS DCRPCER_PROTO DCRPCER_REPLY_TOO_BIG DCRPCER_NO_SUCH_SERVICE DCRPCER_SERVICE_CLOSED DCRPCER_SYSERR_AT_SERVER DCRPCER_NO_BUFS_AT_SERVER DCRPCER_ALL_RECEIVED	00301 00302[#1, #6] 00309 00311 00312 00316 00318 00321
	dc_rpc_discard_further_replies [CBLDCRPC(DISCARDF)]	--	--
	dc_rpc_get_callers_address [CBLDCRPC(GETCLADR)]	--	--
	dc_rpc_set_service_prio [CBLDCRPC(SETSVPRI)]	--	--
	dc_rpc_get_service_prio [CBLDCRPC(GETSVPRI)]	DCRPCER_PROTO	00302[#1]
	dc_rpc_set_watch_time [CBLDCRPC(SETWATCH)]	DCRPCER_INVALID_ARGS DCRPCER_PROTO	00301 00302[#1]
	dc_rpc_get_watch_time [CBLDCRPC(GETWATCH)]	DCRPCER_PROTO	00302[#1]
TAM file service (tam)	dc_tam_close	DCTAMER_PARAM_FLG DCTAMER_PROTO DCTAMER_NOOPEN	-- --[#1] --

Type	Function name [prog_name (request_code)] <DML>	C return value	COBOL return code
	dc_tam_delete [CBLDCTAM(ERS or ERSR)]	DCTAMER_PARAM_KEY DCTAMER_PARAM_KNO DCTAMER_PARAM_BFA DCTAMER_PARAM_BFS DCTAMER_PARAM_FLG DCTAMER_PROTO DCTAMER_NOOPEN DCTAMER_NOREC DCTAMER_LOCK DCTAMER_MEMORY DCTAMER_IO	01702 01703 01704 01705 01708 01721#1 01726 01731 01736 01769 01770
	dc_tam_get_inf [CBLDCTAM(GST)]	DCTAMER_PARAM_TBL DCTAMER_PARAM_FLG DCTAMER_UNDEF DCTAMER_PROTO	01702 01708 01710 01721#1
	dc_tam_open	DCTAMER_PARAM_TBL DCTAMER_PARAM_FLG DCTAMER_UNDEF DCTAMER_PROTO DCTAMER_NOLOAD DCTAMER_OPENED DCTAMER_LOCK DCTAMER_OPENNUM DCTAMER_IO	-- -- -- --#1 -- -- -- -- --
	dc_tam_read [CBLDCTAM(FxxR or FxxU)]	DCTAMER_PARAM_KEY DCTAMER_PARAM_KNO DCTAMER_PARAM_BFA DCTAMER_PARAM_BFS DCTAMER_PARAM_FLG DCTAMER_PROTO DCTAMER_NOOPEN DCTAMER_IDXTYP DCTAMER_NOREC DCTAMER_LOCK DCTAMER_MEMORY DCTAMER_IO	01702 01703 01704 01705 01708 01721#1 01726 01729 01731 01736 01769 01770

Type	Function name [prog_name (request_code)] <DML>	C return value	COBOL return code
	dc_tam_read_cancel	DCTAMER_PARAM_KEY DCTAMER_PARAM_KNO DCTAMER_PARAM_FLG DCTAMER_PROTO DCTAMER_NOOPEN DCTAMER_NOREC DCTAMER_MEMORY	-- -- -- --#1 -- -- --
	dc_tam_rewrite	DCTAMER_PARAM_KEY DCTAMER_PARAM_KNO DCTAMER_PARAM_DTA DCTAMER_PARAM_DTS DCTAMER_PARAM_FLG DCTAMER_PROTO DCTAMER_NOOPEN DCTAMER_NOREC DCTAMER_MEMORY DCTAMER_IO	-- -- -- -- -- --#1 -- -- -- --
	dc_tam_write [CBLDCTAM(MFY, MFYS, or STR)]	DCTAMER_PARAM_KEY DCTAMER_PARAM_KNO DCTAMER_PARAM_DTA DCTAMER_PARAM_DTS DCTAMER_PARAM_FLG DCTAMER_PROTO DCTAMER_NOOPEN DCTAMER_NOREC DCTAMER_EXKEY DCTAMER_LOCK DCTAMER_NOAREA DCTAMER_MEMORY DCTAMER_IO	01702 01703 01706 01707 01708 01721#1 01726 01731 01735 01736 01763 01769 01770
Transaction control (trn)	dc_trn_begin [CBLDCTRN(BEGIN)]	DCTRNER_PROTO	00905#1
	dc_trn_chained_commit [CBLDCTRN(C-COMMIT)]	DCTRNER_PROTO	00905#1
	dc_trn_chained_rollback [CBLDCTRN(C-ROLL)]	DCTRNER_PROTO	00905#1
	dc_trn_info [CBLDCTRN(INFO)]	1	00001 00908

Type	Function name [prog_name (request_code)] <DML>	C return value	COBOL return code
	dc_trn_unchained_commit [CBLDCTRN(U-COMMIT)]	DCTRNER_PROTO	00905[#1]
	dc_trn_unchained_rollback [CBLDCTRN(U-ROLL)]	DCTRNER_PROTO	00905[#1]
TX interface (tx_~)	tx_begin [TXBEGIN]	TX_PROTOCOL_ERROR	TX_PROTOCO L_ERROR[#1]
	tx_close [TXCLOSE]	--	--
	tx_commit [TXCOMMIT]	TX_PROTOCOL_ERROR	TX_PROTOCO L_ERROR[#1]
	tx_info [TXINFORM]	TX_PROTOCOL_ERROR	TX_PROTOCO L_ERROR[#1]
	tx_open [TXOPEN]	TX_ERROR	TX_ERROR[#1]
	tx_set_commit_return [TXSETCOMMITRET]	TX_EINVAL TX_NOT_SUPPORTED TX_PROTOCOL_ERROR	TX_EINVAL TX_NOT_SUP PORTED TX_PROTOCO L_ERROR[#1]
	tx_set_transaction_control [TXSETTRANCTL]	TX_EINVAL TX_PROTOCOL_ERROR	TX_EINVAL TX_PROTOCO L_ERROR[#1]
	tx_set_transaction_timeout [TXSETTIMEOUT]	TX_EINVAL TX_PROTOCOL_ERROR	TX_EINVAL TX_PROTOCO L_ERROR[#1]
	tx_rollback TXROLLBACK]	TX_PROTOCOL_ERROR	TX_PROTOCO L_ERROR[#1]
XATMI interface (tp~)	tpalloc	TPEINVAL TPENOENT TPESYSTEM TPEPROTO	-- -- -- --[#1]
	tpfree	--	--

Type	Function name [prog_name (request_code)] <DML>	C return value	COBOL return code
	tprealloc	TPEINVAL TPESYSTEM TPEPROTO	-- -- --#1
	tptypes	TPEINVAL TPEPROTO	-- --#1
	tpreturn	--	--
	tpadvertise	TPEINVAL TPEPROTO	-- --#1
	tpunadvertise	TPEINVAL TPEPROTO	-- --#1
	tpacall	TPEINVAL TPEPROTO TPENOENT TPEITYPE TPETRAN	-- --#1, #7, #8, #9 -- -- --
	tpcall	TPEINVAL TPEPROTO TPENOENT TPEITYPE TPEOTYPE TPETRAN TPESVCFAIL TPESVCERR	-- --#1, #7, #8, #9 -- -- -- -- -- --
	tpcancel	TPEBADDESC TPETRAN TPEPROTO	-- -- --#1, #7, #8,
	tpgetrply	TPEBADDESC TPEOTYPE TPESYSTEM TPEPROTO TPESVCFAIL TPESVCERR	-- -- -- --#1, #7, #8, #9, #10 -- --

Type	Function name [prog_name (request_code)] <DML>	C return value	COBOL return code
	tpconnect	TPEINVAL TPENOENT TPEITYPE TPETRAN TPEPROTO	-- -- -- -- -- #1, #7, #8, #9
	tpdiscon	TPEBADDESC TPEPROTO	-- -- #1, #7, #8, #11
	tprecv	TPEINVAL TPEOTYPE TPEBADDESC TPEPROTO	-- -- -- -- #1, #7, #8, #12
	tpsend	TPEINVAL TPEBADDESC TPEPROTO	-- -- -- #1, #7, #8, #13
Online tester (uto)	dc_uto_test_status [CBLDCUTO(T-STATUS)]	DCUTOER_PROTO DCUTOER_PARAM_FLAGS DCUTOER_PARAM_ADDS	02701 #1 02757 02758

Legend:

--: No return value (return code)

Note

For the XATMI interface, the return value in C indicates the value to be returned to tperrno.

#1: If no dc_rpc_open function has been issued.

#2: If the dc_adm_get_nd_status_begin, dc_adm_get_sv_status_begin, or dc_adm_get_nodeconf_begin function has been issued.

#3: If no dc_adm_get_nd_status_begin, dc_adm_get_sv_status_begin, or dc_adm_get_nodeconf_begin function has been issued.

#4: If issued in the main function.

#5: If the dc_mcf_mainloop or dc_rpc_mainloop function has been issued.

#6: If no asynchronous dc_rpc_call function has been issued.

#7: If issued after the tpreturn function.

#8: If issued in a service environment with different service paradigms.

#9: For recursive calls in a service group.

#10: If no tpacall function has been issued.

#11: If not the connection originator.

#12: If the connection attribute is `TPSENDONLY`.

#13: If the connection attribute is `TPRECVONLY`.

14.3 List of functions not supported by the simulation feature

As shown in *14.1(1) Simulation functions*, you can simulate functions provided by OpenTP1 by using the simulation functions of the offline tester. However, functions provided by OpenTP1 that are listed in the following tables are not supported by the simulation functions of the offline tester. Therefore, if these functions are executed by a UAP, only the return values listed in the following tables are returned, and trace information is not acquired nor are function arguments changed. In addition, you cannot set return values in the function return value file.

The following tables separately list the simulation functions not supported for C and for COBOL.

Table 14-3: List of functions not supported by the simulation feature (for C)

Type	Function name	Description of the OpenTP1-provided function	Return value
Remote procedure call (`rpc`)	`dc_rpc_call_to` function	Calls a remote service by specifying the communication destination.	`0`
	`dc_rpc_get_error_descriptor` function	Acquires the descriptor of the asynchronous response RPC request where an error occurred.	`1`
	`dc_rpc_discard_specific_reply` function	Rejects the reception of specific processing results.	`DC_OK`
	`dc_rpc_service_retry` function	Retries a service function.	`DC_OK`
	`dc_rpc_get_gateway_address` function	Acquires the gateway node address.	`DC_OK`
	`dc_rpc_cltsend` function	One-way communication to the CUP	`DC_OK`
Remote API facility (`rap`)	`dc_rap_connect` function	Establishes a connection with a RAP-processing listener.	`DC_OK`
	`dc_rap_disconnect` function	Releases the connection with a RAP-processing listener.	`DC_OK`
Performance verification trace (`prf`)	`dc_prf_utrace_put` function	Acquires the user-specific performance verification trace information.	`DC_OK`
	`dc_prf_get_trace_num` function	Reports the sequential number of the acquired performance verification trace information.	`DC_OK`

Type	Function name	Description of the OpenTP1-provided function	Return value
Message transmission (mcf)	dc_mcf_ap_info function	Reports application information.	DCMCFRTN_00000
	dc_mcf_ap_info_uoc function	Reports application information to a user exit routine.	DCMCFRTN_00000
	dc_mcf_timer_set function	Sets user timer monitoring.	DC_OK
	dc_mcf_timer_cancel function	Cancels user timer monitoring.	DC_OK
DAM file service (dam)	dc_dam_bseek function	Searches for a physical file block.	Returns the relative block number specified in the argument of the function.
	dc_dam_dget function	Directly reads a block from a physical file.	504
	dc_dam_dput function	Directly writes data to a block in a physical file.	504

Table 14-4: List of functions not supported by the simulation feature (for COBOL)

Type	Program name (request code)	Description of the OpenTP1-provided function	Status code
Remote procedure call (rpc)	CBLDCRPC ('GETERDES')	Acquires the descriptor of the asynchronous response RPC request where an error occurred.	00000
	CBLDCRPC ('DISCARDS')	Rejects the reception of specific processing requests.	00000
	CBLDCRPC ('SVRETRY')	Retries a service program.	00000
	CBLDCRPC ('GETGWADR')	Acquires the gateway node address.	00000

Type	Program name (request code)	Description of the OpenTP1-provided function	Status code
Remote API facility (rap)	CBLDCRAP ('CONNECT')	Establishes a connection with a RAP-processing listener.	00000
	CBLDCRAP ('DISCNCT')	Releases the connection with a RAP-processing listener.	00000
Edition of journal data (jnl)	CBLDCJUP ('CLOSERPT')	Closes the jnlrput output file.	00000
	CBLDCJUP ('OPENRPT')	Opens the jnlrput output file.	00000
	CBLDCJUP ('RDGETRPT')	Enters journal data from the jnlrput output file.	00000
Performance verification trace (prf)	CBLDCPRF ('PRFPUT')	Acquires the user-specific performance verification trace information.	00000
	CBLDCPRF ('PRFGETN')	Reports the sequential number of the acquired performance verification trace information.	00000
Transmission of messages (mcf)	CBLDCMCF ('APINFO')	Reports application information.	00000
DAM file service (dam)	CBLDCDMB ('BSEK')	Searches for a physical file block.	00000
	CBLDCDMB ('DGET')	Directly reads a block from a physical file.	00000
	CBLDCDMB ('DPUT')	Directly writes data to a block in a physical file.	00000

Type	Program name (request code)	Description of the OpenTP1-provided function	Status code
XATMI interface (tp~)	TPCALL	Calls a request or response service and receives the reply.	TPOK#
	TPACALL	Calls a request or response service.	TPOK#
	TPGETRPLY	Receives an asynchronous reply from a request or response service.	TPOK#
	TPCANCEL	Cancels a request or response service.	TPOK#
	TPCONNECT	Establishes a connection with an interactive service.	TPOK#
	TPDISCON	Disconnects an interactive service.	TPOK#
	TPRECV	Receives a message from an interactive service.	TPOK#
	TPSEND	Sends a message to an interactive service.	TPOK#
	TPADVERTISE	Advertises a service name.	TPOK#
	TPUNADVERTISE	Cancels the advertisement of a service name.	TPOK#
	TPSVCSTART	Starts a service routine.	TPOK#
	TPRETURN	Returns control from a service routine.	There is no status code.

#: TPOK is set in the data area (TP-STATUS) where a return value indicating the result of execution is set.

Chapter
15. How to Use UAP Traces

This chapter describes how to use UAP traces.

This chapter contains the following sections:

15.1 Collecting UAP traces
15.2 Editing and outputting UAP traces

15.1 Collecting UAP traces

The *UAP trace facility* collects information on the OpenTP1 functions called from a UAP. OpenTP1 collects UAP traces in UAP trace data files and in process-specific areas.

If a UAP terminates abnormally, the user can edit and output a log file of the library functions called from the UAP and analyze why the UAP terminated abnormally.

The UAP trace facility can be used for the following UAP events:

- Abnormal termination of a UAP
- Forcible termination of a UAP by the `dcstop -df` command
- Forcible termination of a UAP by the `dcsvstop -df` command
- Forcible termination of a UAP by the `prckill` command

15.1.1 UAP trace collection units

UAP traces are collected separately for each UAP process. The UAP traces are edited and output based on either the UAP trace data file or the core file collected for each UAP process.

UAP traces are collected for SUPs, SPPs, and MHPs.

15.1.2 Trace area definition

The size of the area used by the UAP trace facility is specified using the `uap_trace_max` operand in the user service definition.

See the manual *OpenTP1 System Definition* for details on the user service definition.

15.1.3 Information to collect

A UAP trace contains various information specified for arguments when the UAP calls OpenTP1 library functions. Of this information, the *exit information* from functions maintains information when a function returned. The *entry information* to functions maintains information when a function call from the UAP caused an entry into the OpenTP1 function.

When the online tester (TP1/Online Tester) is used, UAP trace data contains entry information and exit information for all executed functions.

When collecting of the complete I/O data is specified with the online tester (TP1/Online Tester) used, I/O data is also collected.

15.2 Editing and outputting UAP traces

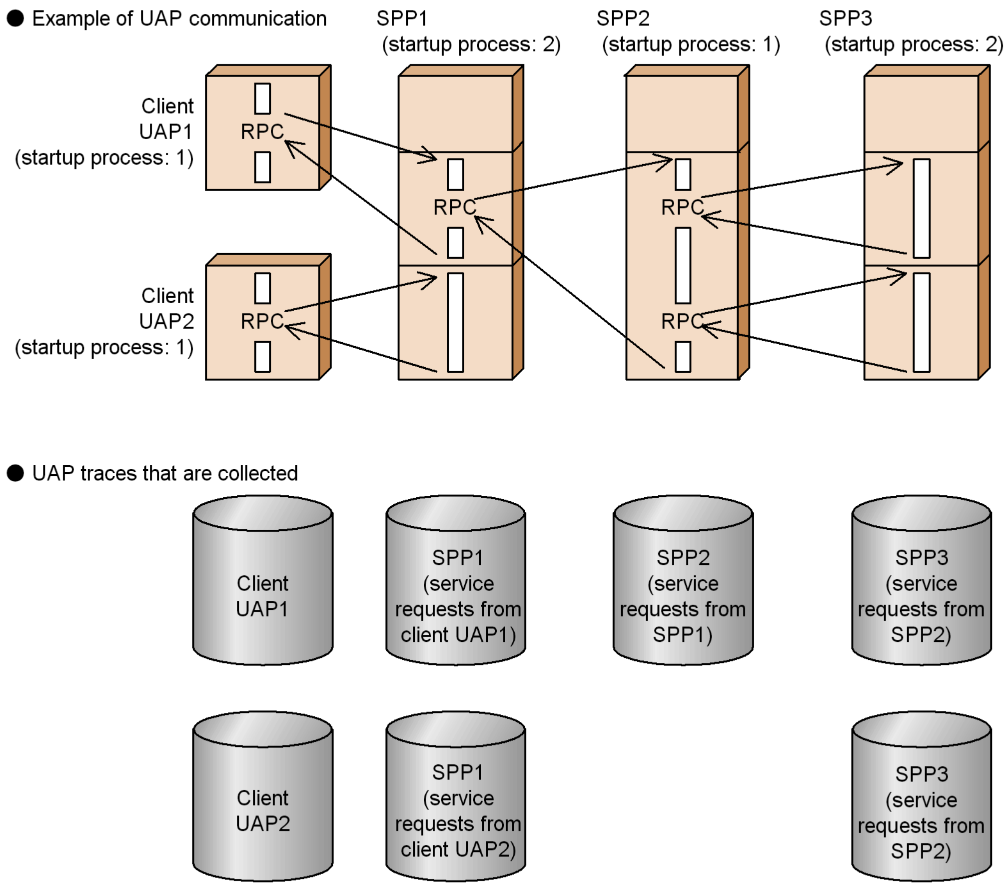
The following explains how to edit and output UAP traces.

15.2.1 UAP trace output units

UAP traces are edited and output by process unit. When two or more processes are involved in a transaction, traces are output only information of transaction branch that executed at the UAP that terminated abnormally.

The example in Figure 15-1 shows communication among UAPs and the UAP traces collected.

Figure 15-1: Inter-UAP communication and collected UAP traces



15.2.2 UAP trace output methods

There are the following two methods of editing and outputting UAP traces.

(1) Edit and output the trace to a file automatically

The file that stores abnormal termination information that OpenTP1 collects for each UAP process is called a *core file*. If UAP abnormally terminates and there is a core file, the UAP trace is automatically edited and output to a file called the *UAP trace output file*.

Table 15-1 shows the directories and file names of the core file and UAP trace output file.

Table 15-1: Directories and file names of core file and UAP trace output file

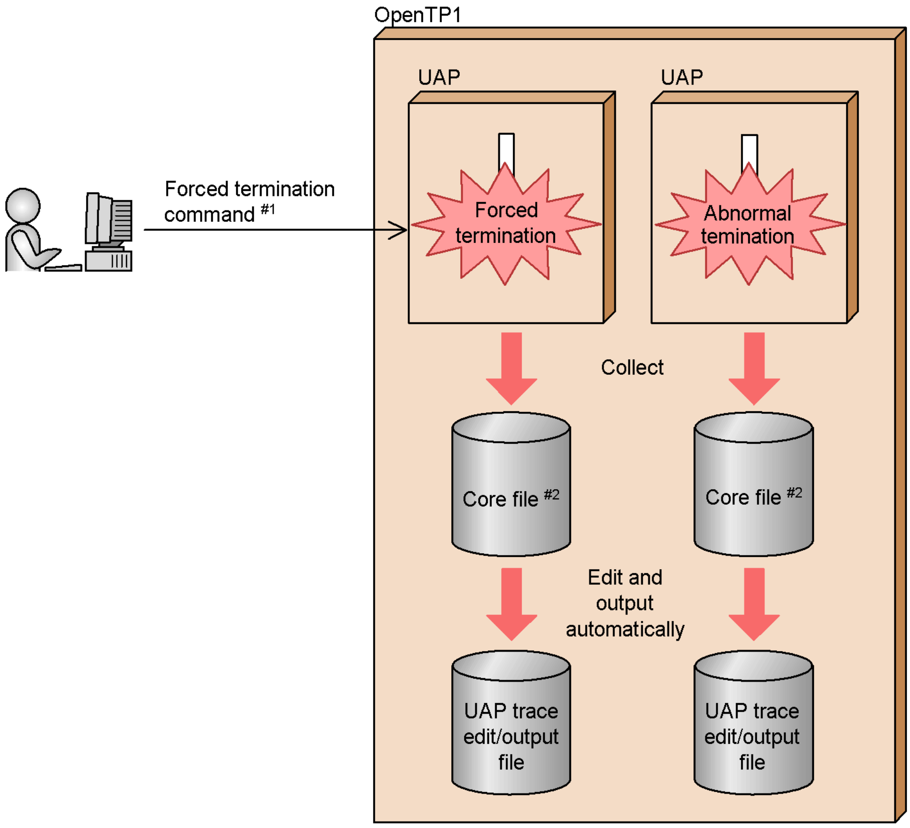
Name	Directory	File name
Core file	`$DCDIR/spool/save/`	*server-name-n*#
UAP trace output file	`$DCDIR/spool/save/`	*server-name-n*`.uat`#

#: *n*: Sequence number of the core file (1 to 3)

Note that a sequence number is not assigned to the core file output if OpenTP1 is forcibly terminated (when the `dcsvstop -df` command is executed or the real monitoring time expires).

Figure 15-2 shows an overview of automatically editing and outputting a UAP trace to a file.

Figure 15-2: Overview of automatic edit and output of UAP trace



#1
Refers to any of the following commands:

- `dcsvstop -df` command
- `prckill` command
- `dcstop -df` command

#2
If `Y` is specified for the `uap_trace_file_put` operand, a UAP trace data file is automatically edited and output, instead of a core file.
The `uap_trace_file_put` operand can be specified in any of the following definitions:

- System common definition
- User service default definition
- User service definition

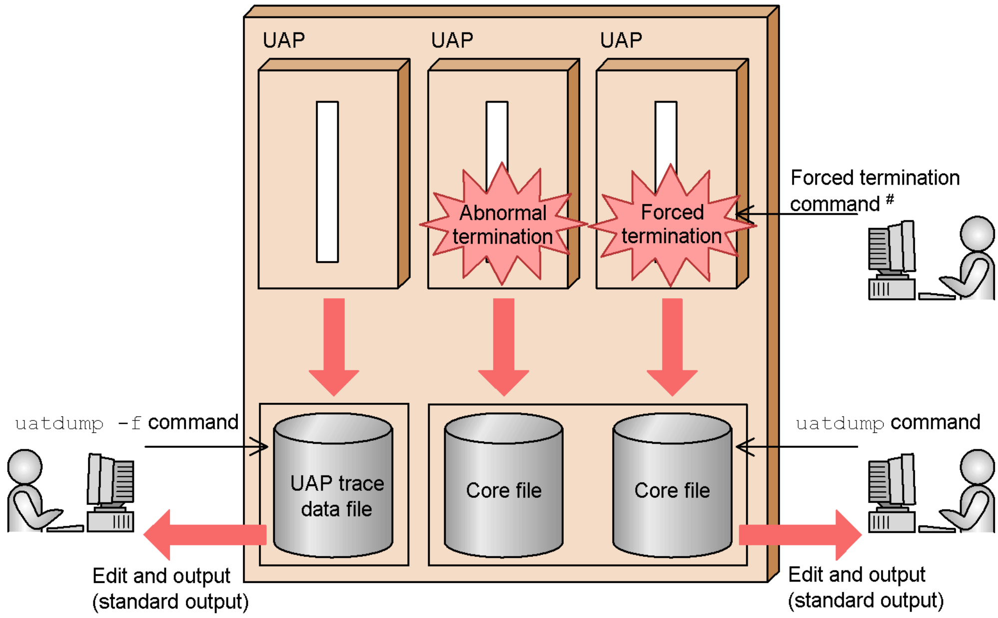
(2) Edit and output the trace to the standard output by a command

When the `uatdump` command is entered, the UAP trace is edited and output to the standard output. For details on how to use the `uatdump` command, see Subsection

15.2.3 uatdump (edited output of UAP trace).

Figure 15-3 shows an overview of editing and outputting the UAP trace to the standard output by a command.

Figure 15-3: Overview of editing and outputting UAP trace to standard output by a command



#

Refers to any of the following commands:

- `dcsvstop -df` command
- `prckill` command
- `dcstop -df` command

15.2.3 uatdump (edited output of UAP trace)

(1) Syntax

```
uatdump {[core-file-name] | -f [UAP-trace-data-file]}
```

(2) Function

Edits a specified UAP trace data file or core file, and outputs the contents to standard output.

On a node that uses the online tester (TP1/Online Tester), this command edits and outputs exit information and entry information for all executed functions. Since the command does not output tester information, however, some data may be missing just after the tester information.

(3) Options

- `-f` *UAP-trace-data-file* ~<pathname>

  Specify the pathname of the UAP trace data file to which UAP traces are edited and output.

  If specification of this argument is omitted, `ducat.map` in the current command execution directory is assumed for the UAP trace data file name.

(4) Command argument

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

  Specify the pathname of the core file for the UAP process that terminated abnormally.

  If specification of this argument is omitted, `core` in the current command execution directory is assumed for the core file name.

(5) Output messages

Message ID	message text	Output file
KFCA03100-E	Insufficient memory.	Standard error output
KFCA03101-E	Invalid option flag.	Standard error output
KFCA03102-E	Specified file does not exist.	Standard error output
KFCA03103-E	No trace data in the specified file.	Standard error output
KFCA03104-W	Incorrect type code in the trace data.	Standard error output
KFCA03105-I	Help message	Standard output

(6) Output format

See *15.2.4 UAP trace output format* for the output format of the `uatdump` command.

(7) Notes

Always specify the `-f` option when a UAP trace data file is being edited and output if `Y` is specified for the `uap_trace_file_put` operand. If you do not specify the `-f` option, the command ends in an error because the UAP traces cannot be edited.

The `uap_trace_file_put` operand is specified in one of the following definitions:

- System common definition
- User service default definition

- User service definition

15.2.4 UAP trace output format

The following shows the format of the UAP trace automatically edited and output to a file and the format of the UAP trace edited and output to the standard output by using the `uatdump` command.

(1) Output format

```
HIUXOLTF[HIUXOLTF]                                                     1.
SERVICE GROUP NAME = dam_svg
PROCESS ID = 2029                          SIZE =8576

FUNCTION = dc_rpc_open (EXIT)                                      2.   3.
    COLLECTION DATE AND TIME = 98/07/20 11:39:20
    COLLECTION NO. = 1      SERVICE NAME = ****
      SERVER NAME = damspp
      OPTION FLAG = 0x00000000 (DCNOFLAGS)
    RETURN CODE = 0  (NORMAL TERMINATION)

FUNCTION = dc_dam_open (EXIT)                                      2.
    COLLECTION DATE AND TIME = 98/07/20 11:39:20
    COLLECTION NO. = 2      SERVICE NAME = ****
      REQUEST CODE = OPEN
      LOGICAL FILE NAME = sppfile
      OPTION FLAG = 0x00000022 (DCDAM_NOWAIT)
                              (DCDAM_BLOCK_EXCLUSIVE)
    RETURN CODE = 0  (FILE DESCRIPTOR)

     :     :     :     :     :     :     :     :     :
     :     :     :     :     :     :     :     :     :
```

Legend:

1. UAP trace header
2. UAP trace data

   When the online tester is used, entrance information and exit information are output alternately. `ENTRANCE` and `EXIT` are displayed, accordingly.
3. Output area for the call information on the OpenTP1 function.

   The information that is output to the output area depends on the function that is issued.

Explanation:

`SERVICE GROUP NAME`

Service group name of the active service.

Asterisks (`****`) are displayed for a SUP or MHP.

`PROCESS ID`

Process ID of the process for which the UAP trace was collected

`SIZE`

Size of the UAP trace information area (decimal display; bytes)

`FUNCTION`

Called OpenTP1 function

`COLLECTION DATE AND TIME`

Date and time of collection (last 2 digits of year/month/day hour:minute:second)

`COLLECTION NO.`

Sequential number set when the UAP trace data was collected (up to 6 digits)

`SERVICE NAME`

Active service name (up to 32 characters).

Asterisks (`****`) are displayed for a SUP or MHP.

`RETURN CODE`

Execution result of the OpenTP1 function

(2) Output example

```
HIUXOLTF[HIUXOLTF]
SERVICE GROUP NAME = dam_svg
PROCESS ID = 2029                          SIZE =8576


FUNCTION = dc_rpc_open (EXIT)
    COLLECTION DATE AND TIME = 98/07/20 11:39:20
    COLLECTION NO. = 2      SERVICE NAME = ****
      SERVER NAME = damspp
      OPTION FLAG = 0x00000000 (DCNOFLAGS)
    RETURN CODE = 0  (NORMAL TERMINATION)

FUNCTION = dc_dam_open (EXIT)
    COLLECTION DATE AND TIME = 98/07/20 11:39:20
    COLLECTION NO. = 3      SERVICE NAME = ****
      REQUEST CODE = OPEN
      LOGICAL FILE NAME = sppfile
      OPTION FLAG = 0x00000022 (DCDAM_NOWAIT)
                               (DCDAM_BLOCK_EXCLUSIVE)
    RETURN CODE = 0  (FILE DESCRIPTOR)

FUNCTION = XATMI STARTING FUNCTION (EXIT)
    COLLECTION DATE AND TIME = 98/07/20 11:39:25
    COLLECTION NO. = 4      SERVICE NAME = REFSVC_A
      NODE NAME = 2c3gfm01
      SERVICE NAME = REFSVC_A
      RECEIVE TYPE NAME = X_C_TYPE
      RECEIVE SUBTYPE NAME = sub1
      ------ RECEIVE DATA ------
      00008c    534b492d 50415241 44494345 00000000   SKI- PARA DICE ....
      00009c    00000000 00000000 00000000 00000000   .... .... .... ....
      RECEIVE DATA LENGTH = 104
      FLAG = 0x00000000
      DESCRIPTOR = 0

FUNCTION = tpconnect (EXIT)
    COLLECTION DATE AND TIME = 98/07/20 11:39:20
    COLLECTION NO. = 5      SERVICE NAME = REFSVC_A
      SERVICE NAME = REFSVC_C
      SEND TYPE NAME = X_C_TYPE
      SEND SUBTYPE NAME = sub1
      ------ SEND DATA ------
      00006c    534b492d 50415241 44494345 00000000   SKI- PARA DICE ....
      00007c    00000000 00000000 00000000 00000000   .... .... .... ....
      SEND DATA LENGTH = 50
      FLAG = 0x00001000 (TPRECVONLY)
    RETURN CODE = 1390287197
```

```
FUNCTION = tprecv (EXIT)
    COLLECTION DATE AND TIME = 98/07/20 11:39:25
    COLLECTION NO. = 6      SERVICE NAME = REFSVC_A
      DESCRIPTOR = 1390287197
      RECEIVE TYPE NAME = X_C_TYPE
      RECEIVE SUBTYPE NAME = sub1
      ------ RECEIVE DATA ------
      000050    4e4f5254 482d464c 49474854 00000000   NORT H-FL IGHT ....
      000060    00000000 00000000 00000000 00000000   .... .... .... ....
      RECEIVE DATA LENGTH = 104
      FLAG = 0x00000000
      EVENT = 0x0008 (TPEV_SVCSUCC)
      tperrno = 22 (TPEEVENT)
      tpurcode = 0x00000000
    RETURN CODE = -1

FUNCTION = tpreturn (EXIT)
    COLLECTION DATE AND TIME = 98/07/20 11:39:25
    COLLECTION NO. = 7      SERVICE NAME = REFSVC_A
      RETURN VALUE = 0x04000000            USER RETURN = 22
      SEND TYPE NAME = X_C_TYPE
      SEND SUBTYPE NAME = sub1
      ------ SEND DATA ------
      000054    4e4f5254 482d464c 49474854 00000000   NORT H-FL IGHT ....
      000064    00000000 00000000 00000000 00000000   .... .... .... ....
      SEND DATA LENGTH = 50
      FLAG = 0x00000000

FUNCTION = XATMI ENDING FUNCTION (EXIT)
    COLLECTION DATE AND TIME = 98/07/20 11:39:25
    COLLECTION NO. = 8      SERVICE NAME = REFSVC_A
      NODE NAME = 2c3gfm01
      SERVICE NAME = REFSVC_A
      SEND TYPE NAME = X_C_TYPE
      SEND SUBTYPE NAME = sub1
      ------ SEND DATA ------
      00008c    4e4f5254 482d464c 49474854 00000000   NORT H-FL IGHT ....
      00009c    00000000 00000000 00000000 00000000   .... .... .... ....
      SEND DATA LENGTH = 104
```

```
FUNCTION = XATMI STARTING FUNCTION (EXIT)
    COLLECTION DATE AND TIME = 98/07/20 11:39:25
    COLLECTION NO. = 9      SERVICE NAME = REFSVC_B
      NODE NAME = 2c3gfm01
      SERVICE NAME = REFSVC_B
      RECEIVE TYPE NAME = X_C_TYPE
      RECEIVE SUBTYPE NAME = sub1
      ------ RECEIVE DATA ------
      00008c    534b492d 50415241 44494345 00000000   SKI- PARA DICE ....
      00009c    00000000 00000000 00000000 00000000   .... .... .... ....
      RECEIVE DATA LENGTH = 104
      FLAG = 0x00000c00 (TPSENDONLY)
                        (TPCONV)
      DESCRIPTOR = 1028921693

     :     :     :     :     :     :     :     :     :
     :     :     :     :     :     :     :     :     :
```

Index

A

abbreviations for products iv
acronyms ix
application program startup requests, simulating 25
application startup messages, invalidating 161
application test
- starting 184
- terminating 187

application, testing 166
asynchronous receive message file 71

C

call 302
client UAP
- simulating 13, 197
- simulating, with RPC interface 13, 102, 197
- simulating, with TxRPC interface 198
- simulating, with XATMI interface 14, 102, 198

client UAP simulator 13, 102, 197
cmdauto 303
comment statement 85, 260
complete I/O data trace, collecting 34
continuous commands, executing 209, 281, 303
continuous execution command file 236
- creating 236
- directory definition for 226

continuous execution commands, setting 236
continuous inquiry responses, simulating 23
conventions
- abbreviations for products iv
- acronyms ix
- diagrams x
- fonts and symbols xi
- KB, MB, GB, and TB xiii
- permitted characters xiii
- version numbers xiv

conversational service paradigm 15
core file 9, 350

D

DAM and TAM files, notes on 288
DAM file 255
DAM service simulator 203
DAM service, simulating 203
DCUTOKEY 59
debugger
- activating UAP interlocked with 113
- interlocking 40
- specifying connection 282
- terminating UAP interlocked with 112

debugger connection 210
definition
- offline tester environment 214
- system service configuration 44
- tester service 44
- user service 48, 231

diagram conventions x
dummy SPP 49

E

end 304
end statement 86, 261
entry information 348
environment variables, setting 59
- DCUTOKEY 59
- test user ID 59

environment-var-name 231
error conditions and causes 154
error events, suppressing 161
error recovery 153
- handling online tester errors 154

event type, setting 233
exit information 348

F

facilities [offline tester]
- client UAP simulator 197
- collecting offline tester trace information 211
- continuous command execution 209
- creating tester files 208
- DAM service simulator 203
- debugger connection 210
- MCF simulator 202
- operating command simulator 207
- server UAP simulator 199
- TAM service simulator 204

facilities [online tester] 12, 31
- client UAP simulator 13
- collecting complete I/O data trace 34
- collecting UAP trace information 34
- creating tester file 31
- debugger interlocking 40
- disabling resource updating 28
- editing send messages 39
- MCF simulator 22
- operating command simulator 29
- server UAP simulator 18
- tester file edit and output 33

file automatically, editing and outputting trace to 350
file errors 156
file service, simulating
- DAM service simulator 203
- TAM service simulator 204

files created by offline tester
- list of 270
- temporary memory data file 270
- trace file 270
- XATMI send data file 270

files created by online tester
- MCF send message file 96
- service response data file 95
- temporary memory data file 96
- trace file 96
- XATMI send data file 96

files created by online tester, list of 95
files created by user 239
font conventions xi
function return values
- event type, setting 233
- output data, setting 234
- return value, setting 233
- setting 232

function return values file 232
- creating 232
- definition of 229

functions not supported by simulation feature 342

G

GB meaning xiii

I

information to collect 348
- entry information 348
- exit information 348

input data definition statement 87, 261
interface definition language file 238
internode shared table definitions 228

K

KB meaning xiii

L

logical terminal information, specifying 47
logical terminal test
- starting 178
- terminating 180

logical terminal, testing 166

M

max_message_file_size 46
max_trace_file_size 45
MB meaning xiii
MCF
- editing send messages 39
- simulating 22, 202
- simulating application program startup requests 25
- simulating continuous inquiry responses 23
- simulating message send/receive 22
- simulating synchronous point processing 27
- simulation functions 22

MCF online tester 6
- collecting test information 164
- collecting UAP trace information 164, 170
- displaying test mode information 170
- editing test information 170
- inheriting test mode information 169
- merging and outputting UAP trace information 170
- MHP testing 160
- starting and ending test 166
- starting test 166
- test environment 166
- test mode 166
- test mode information 166
- test mode messages 167
- test mode range 167

MCF online tester status, displaying 174
MCF online tester use declaration 174
MCF receive message file, directory definition for 225
MCF receive message files 71, 250
- asynchronous receive message file 71
- synchronous receive message file 76

MCF send message file 96
MCF simulation functions, UAP traces for 109
MCF simulator 22, 202
mcfauape 187
mcfauaps 184
mcfaulsap 181
mcflsutf 174
mcftulee 180
mcftules 178
mcftulsle 176
mcftulssg 189
mcftusge 193
mcftusgs 191
mcfutfst 174
message send/receive, simulating 22
MHP automatic shutdown, suppressing 162
MHP testing 160
- disabling non-MCF resources update 160
- invalidating application startup messages 161
- invalidating send messages 160
- suppressing error events 161
- suppressing MHP automatic shutdown 162

MHP, service requests to 103, 130

N

non-MCF resources, disabling update of 160
non-test UAP 49
notes on
- DAM and TAM files 288
- offline tester 284
- running tests 284
- UAP 289

O

offline test
- ending 277
- starting 277

offline tester 2, 6
- creating stubs 238
- creating tester files 280
- creating UAP 272
- creating UAP execution format programs 272
- executing continuous commands 281
- facilities of 196
- files created by 270
- files created by user 239
- inputting tester file name to 305, 308
- list of simulation functions and processing 310
- notes on 284
- requesting service 279
- setting continuous execution commands 236
- setting function return values 232
- specifying debugger connection 282
- starting 293
- system definitions for 214
- terminating 304
- test data definition file 259
- test data definition file, creating 259
- TP1/Offline Tester 2, 6
- user service definition 231

offline tester DAM file, creating 292
offline tester environment definition 214
- continuous execution command file, directory definition for 226

DAM file definitions 227
function return values file, definition of 229
internode shared table definitions 228
MCF receive message file, directory definition for 225
operating command result data file, directory definition for 225
protocol definition 230
RPC request data file, directory definition for 221
RPC response data file, directory definition for 223
TAM file definitions 227
trace file definition 229
TxRPC request data file, directory definition for 222
TxRPC response data file, directory definition for 224
UAP definition 219
XATMI request data file, directory definition for 222
XATMI response data file, directory definition for 223
XATMI send/receive data file, directory definition for 224
offline tester TAM files, creating 295
offline tester trace information
collecting 211
editing 283
online tester 2, 3
creating tester file 31
facilities of 12
files created by 95
service response data file 95
system definitions for 44
TP1/Message Control 2, 6
TP1/Message Control/Tester 2, 6
TP1/Online Tester 3
TP1/online tester 2
TP1/Server Base 2, 3
trace file 96
online tester errors
conditions and causes of 154
handling 154, 155
handling UAP errors 156
occur in file 156
OpenTP1 functions, simulating 206
simulation functions 206
operating command output data, creating tester files using 105
operating command result data file 81, 257
directory definition for 225
operating command simulator 29, 207
operating commands 111, 173, 291
activating UAP interlocked with debugger 113
creating offline tester DAM file 292
creating offline tester TAM file 295
creating tester file 115, 293
displaying MCF online tester status 174
displaying test mode information for application 181
displaying test mode information for logical terminal 176
displaying test mode information for service group 189
displaying test status 129
editing and outputting send messages 131
editing and outputting test file content 116
editing and outputting UAP trace information 138
for running tests 112, 174, 292
for testing application 181
for testing logical terminal 176
for testing service group 189
MCF online tester use declaration 174
merging UAP trace information 137
requesting service to MHP 130
requesting service to RPC interface SPP 136
requesting service to XATMI interface SPP 150
retrieving offline tester trace information 296
simulating 29, 207
starting application test 184
starting logical terminal test 178
starting offline tester 293
starting service group test 191
terminating application test 187

terminating logical terminal test 180
terminating service group test 193
terminating UAP interlocked with debugger 112
operating commands for running tests 112, 292
mcflsutf 174
mcfutfst 174
utfdamcre 292
utffilcre 293
utfstart 293
utftamcre 295
utftrcpic 296
utodbgstop 112
utodebug 113
utofilcre 115
utofilout 116
utols 129
utomhpsvc 130
utomsgout 131
utosppsvc 136
utotrcmrg 137
utotrcout 138
utoxsppsvc 150
operating commands for testing application 181
mcfauape 187
mcfauaps 184
mcfaulsap 181
operating commands for testing logical terminal 176
mcftulee 180
mcftules 178
mcftulsle 176
operating commands for testing service group 189
mcftulssg 189
mcftusge 193
mcftusgs 191
output data, setting 234

P

permitted character conventions xiii
protocol definition 230
ps 304

R

read 305
recv statement 57
request/response service paradigm 14
resource updating, disabling 28
return values
for simulation functions 326
setting 233
RPC interface
creating UAP execution format program with 272
simulating client UAP with 13, 102, 197
simulating server UAP with 18, 102, 200
RPC interface definition file 238
RPC interface SPP, service requests to 136
RPC request data file 62, 240
directory definition for 221
RPC response data file 65, 95, 244
directory definition for 223
rpc_trace 47
rpc_trace_name 47
rpc_trace_size 47

S

send messages
editing 39
editing and outputting 109
invalidating 160
send statement 56
send/receive control file 56
send/receive procedures, setting 56
recv statement 56
send statement 56
send/receive control file 56
sep statement 86, 261
server UAP
simulating 18, 199
simulating, with RPC Interface 200
simulating, with RPC interface 18, 102
simulating, with TxRPC Interface 200
simulating, with XATMI Interface 200
simulating, with XATMI interface 19, 102
server UAP simulator 18, 102, 199
server_type 231
service 231
service group

activating 306
displaying test mode information for 189
terminating 307
service group test
starting 191
terminating 193
service group, testing 166
service request data files 62, 240
RPC request data file 62, 240
TxRPC request data file 243
XATMI request data file 63, 241
service requests 279, 302
to MHP 103
to SPP 102
service response data file 65, 244
RPC response data file 65, 95, 244
TxRPC response data file 247
XATMI response data file 66, 95, 245
setting
environment variables 59
send/receive procedures 56
test environment 166
typed buffer information 54
simulation feature, functions not supported by 342
simulation functions 206, 309
list of 310
list of return values for 326
SPP, service requests to 102
standard output by command, editing and outputting trace to 351
uatdump 352
start 306
start statement 85, 260
stop 307
stubs, creating 238
subcommands for running tests 302
activating service group 306
call 302
cmdauto 303
displaying test status 304
end 304
executing continuous commands 303
inputting tester file name to offline tester 305, 308
ps 304
read 305
requesting service 302
start 306
stop 307
terminating offline tester 304
terminating service group 307
write 308
symbol conventions xi
synchronous point processing, simulating 27
synchronous receive message file 76
system definitions
for offline tester 214
for online tester 44
system service configuration definition 44
uto_conf 44

T

TAM file 256
TAM file definitions 227
TAM service simulator 204
TAM service, simulating 204
TB meaning xiii
temporary memory data file 96, 270
test
duplicate test mode specifications 168
ending 167
notes on running 284
operating commands for running 174
running 99, 165, 271
setting environment of 166
starting 166
starting and ending 166
subcommands for running 302
testing application 166
testing logical terminal 166
testing service group 166
test data definition file 84, 259
comment statement 85, 260
creating 84, 259
end statement 86, 261
input data definition statement 87, 261
sep statement 86, 261
start statement 85, 260

- using, to create tester files 104

test directory 84
test environment 166
- creating files 239
- setting 43, 166, 213

test information
- checking UAP response data 110
- checking UAP send data 110
- collecting 34, 164, 211
- collecting UAP trace information 107
- displaying test status 107
- editing 107, 170
- editing and outputting send messages 109
- merging and outputting UAP trace information 108

test mode 48, 166
- dummy SPP 49
- non-test UAP 49
- simulate MHP 49
- test-only UAP 49
- usable UAP 49

test mode information 166
- displaying 170
- displaying, for application 181
- displaying, for logical terminal 176
- inheriting 169

test mode messages 167
test mode range 167
test mode specifications, duplicate 168
test status, displaying 107, 304
test user ID 59
test-only UAP 49
test_adm_call_command 51
test_data_trace 52
test_debugger 52
test_mode 48
test_transaction_commit 51
test_xatmi_send_file 52
tester file 31
- creating 31
- creating and outputting 31
- editing 33
- outputting 33

tester file creation facility 208
tester file edit and output facility 33
tester files 208
- creating 104, 105, 208, 280

tester files, creating
- test data definition file 84

tester service definition 44
- command format 47
- max_message_file_size 46
- max_trace_file_size 45
- rpc_trace 47
- rpc_trace_name 47
- rpc_trace_size 47
- specifying logical terminal information 47
- uto_server_count 45
- watch_time 46

tester, overview of 3
TP1/Message Control 6
TP1/Message Control online tester, using 2
TP1/Message Control/Tester 2, 6
TP1/Offline Tester 2, 6
TP1/Online Tester 2, 3
TP1/Server Base 4
TP1/server base online tester, using 2
trace area definition 348
trace file 270
trace file definition 229
trace information
- collecting offline tester 211
- editing offline tester 283
- retrieving offline tester 296

TxRPC interface
- creating UAP execution format program with 274
- simulating client UAP with 198
- simulating server UAP with 200

TxRPC request data file 243
- directory definition for 222

TxRPC response data file 247
- directory definition for 224

typed buffer definition file 54
typed buffer, setting 54
- typed buffer definition file 54

U

UAP
- activating 278
- creating 100, 272
- notes on 289
- terminating 278

UAP definition 219
UAP errors 156
UAP execution format program
- creating 272
- creating, with RPC or XATMI interface 272
- creating, with TxRPC interface 274

UAP response data, checking 110
UAP send data, checking 110
UAP trace collection units 348
UAP trace data file 9
UAP trace information
- collected for MCF simulation functions 109
- collecting 34, 107, 164, 170
- editing 35
- editing and outputting 138
- merging 35, 137
- merging and outputting 108, 170
- outputting 35

UAP trace output file 350
UAP trace output format 354
UAP trace output methods 350
- editing and outputting trace to file automatically 350
- editing and outputting trace to standard output by command 351

UAP traces 9
- collecting 348
- editing 349
- editing and outputting 352
- information to collect 348
- output format of 354
- outputting 349
- overview of 9
- UAP trace output methods 350
- UAP trace output units 349
- using 347

uap_trace_file_put 353
uatdump 352
usable UAP 49
user service definition 48, 231
- environment-var-name 231
- server_type 231
- test_adm_call_command 51
- test_data_trace 52
- test_debugger 52
- test_mode 48
- test_transaction_commit 51
- test_xatmi_send_file 52
- trace area 348

user-created files 60, 239
- DAM file 255
- list of 239
- MCF receive message files 71, 250
- operating command result data file 81, 257
- service request data files 62, 240
- service response data files 65, 244
- TAM file 256
- XATMI receive data file 68, 248

utfdamcre 292
utffilcre 293
utfstart 293
utftamcre 295
utftrcpic 296
uto_conf 44
uto_server_count 45
utodbgstop 112
utodebug 113
utofilcre 115
utofilout 116
utols 129
utomhpsvc 130
utomsgout 131
utosppsvc 136
utoterm 47
utotrcmrg 137
utotrcout 138
utoxsppsvc 150

V

version number conventions xiv

W

watch_time 46
write 308

X

XATMI interface
    conversational service paradigm 15, 20
    creating UAP execution format program with 272
    request/response service paradigm 14, 19
    simulating client UAP with 14, 102, 198
    simulating server UAP with 19, 102, 200
XATMI interface definition file 238
XATMI interface SPP, service requests to 150
XATMI receive data file 68, 248
XATMI request data file 63, 241
    directory definition for 222
XATMI response data file 66, 95, 245
    directory definition for 223
XATMI send data file 96, 270
XATMI send/receive data file, directory definition for 224

Reader's Comment Form

We would appreciate your comments and suggestions on this manual. We will use these comments to improve our manuals. When you send a comment or suggestion, please include the manual name and manual number. You can send your comments by any of the following methods:

- Send email to your local Hitachi representative.
- Send email to the following address:
  WWW-mk@itg.hitachi.co.jp
- If you do not have access to email, please fill out the following information and submit this form to your Hitachi representative:

Manual name:	
Manual number:	
Your name:	
Company or organization:	
Street address:	
Comment:	

(For Hitachi use)